

LAGRANGIAN / EULERIAN NUMERICAL  
METHODS FOR FLUID INTERFACE ADVECTION  
ON UNSTRUCTURED MESHES

Vom Fachbereich Mathematik  
der Technischen Universität Darmstadt  
zur Erlangung des Grades eines Doktors der  
Ingenieurwissenschaften (Dr.-Ing.)  
genehmigte Dissertation  
von  
Tomislav Marić, M.Sc.  
aus Zagreb, Kroatien

Referent: Prof. Dr. rer.nat. Dieter Bothe  
1. Korreferent: Prof. Dr.-Ing. Martin Oberlack  
2. Korreferent: Prof. Hrvoje Jasak, Ph.D.

Tag der Einreichung: 19. 05. 2017  
Tag der mündlichen Prüfung: 07. 11. 2017

Darmstadt, 2017  
D17



---

## ABSTRACT

---

In this thesis two developed Lagrangian / Eulerian numerical are presented for advecting the sharp fluid interface between immiscible fluids: a dimensionally un-split geometrical Volume-of-Fluid method and a coupled Level Set / Front Tracking method. Both numerical methods support solution domains discretized with unstructured meshes.

Different enhancements of the dimensionally un-split geometrical Volume-of-Fluid method are proposed. A new triangulation algorithm for congruent polyhedra is introduced that accurately decomposes polyhedra with non-convex faces into tetrahedra, allowing for a more accurate volume calculation. Additionally, a significant reduction of complexity in the flux contribution calculation is proposed by reducing the number of required intersection operations. A novel simple interface reconstruction algorithm is developed that ensures second-order accuracy of the interface advection. A conservative error redistribution algorithm is developed that supports parallel execution and ensures volume conservation near machine tolerance, numerical stability and exact numerical boundedness of the solution.

Furthermore, for the coupled Level Set / Front Tracking method, an efficient combination of octree and known vicinity search algorithms is proposed, for fast Front Tracking on unstructured meshes. A third-order accurate in time explicit single-step integration method is proposed for the point displacements, along with a second-order accurate interpolation method from cell centers to cell corner points on unstructured meshes, with a low parallel communication overhead.

An efficient and modular software library for 3D geometrical operations in the C++ programming language is developed, that significantly simplifies the development of new transport algorithms. Developed algorithms are extended for parallel computation with the domain decomposition model (using Open MPI), or the hybrid domain decomposition / shared memory parallel programming model (using hybrid Open MPI / OpenMP).



---

## ZUSAMMENFASSUNG

---

In dieser Arbeit werden zwei numerische Methoden für Lagrange-Euler Verfahren zur Advektion einer fluiden Grenzschicht zwischen nicht mischbaren Fluiden entwickelt. Dies ist zum einen ein geometrischer nicht gesplitteter Volume-of-Fluid Ansatz und zum anderen eine gekoppelte Level Set / Front Tracking Methode. Dabei können in beiden Verfahren unstrukturierte Rechengitter verwendet werden.

Zunächst wird ein neuer Zerlegungsalgorithmus für kongruente Polyeder entwickelt, der mit nicht-konvexen Seiten eine präzise Zerlegung des Polyedervolumens erlaubt. Zudem wird durch eine Reduktion notwendiger Schneidungsoperationen die Berechnungskomplexität der Phasenflussbeiträge erheblich verringert.

Mit Hilfe eines neu entwickelten vereinfachten Rekonstruktionsalgorithmus wird eine Fehlerkonvergenz zweiter Ordnung des gesamten Advektionsverfahrens erreicht. Überdies wird ein volumenerhaltender Umverteilungsalgorithmus vorgestellt, der parallelisiert ist, eine Volumenerhaltung nah der Maschinentoleranz sowie eine exakte numerische Beschränkung sicherstellt und die numerische Stabilität der Lösung erlaubt.

Weiterhin wird für die gekoppelte Level Set / Front Tracking Methode eine effiziente Kombination von octree- und distanzbasierten Suchalgorithmen vorgeschlagen, mit dessen Hilfe eine schnelle Evolution der diskretisierten Grenzschicht (der Front) auch auf unstrukturierten Gittern möglich ist. Dazu wird eine Integrationsmethode zur Verschiebung der Gitterpunkte der diskretisierten Grenzschicht beschrieben, die einen Fehler dritter Ordnung in der Zeit aufweist. Zusammen mit einem Interpolationsverfahren zweiter Ordnung im Ort für die Interpolation von Zellzentren auf zugehörige Rechengitterpunkte des unstrukturierten Gitters, führt dies auf niedrige Kommunikationskosten in einer parallelen Implementierung.

Für alle geometrischen Operationen wurde eine modulare und effiziente Softwarebibliothek in der Programmiersprache C++ entwickelt, die eine Weiterentwicklung neuer Transportalgorithmen wesentlich vereinfacht. Desweiteren wurden die genannten Algorithmen für die parallele Ausführung anhand der Gebietszerlegungsmethode (Open MPI), oder der hybriden Gebietszerlegungsmethode / Shared-Memory-Programmierung (Open MPI / OpenMP) erweitert.



---

## PUBLICATIONS

---

Ideas, figures and text presented in this thesis have appeared previously in the following publications:

T. Marić, H. Marschall, and D. Bothe. voFoam - A geometrical volume of fluid algorithm on arbitrary unstructured meshes with local dynamic adaptive mesh refinement using OpenFOAM. *arXiv preprint arXiv:1305.3417*, 2013

T. Marić, H. Marschall, and D. Bothe. An object-oriented validation library for two-phase DNS algorithms. In *Euromech Colloquium 555 on Small-Scale Numerical Methods for Multi-Phase Flows*, Bordeaux, France, August 28–30, 2013

T. Marić, H. Marschall, and D. Bothe. On the Adaptive Mesh Refinement for a 3D geometrical Volume-of-Fluid transport algorithm on unstructured meshes using OpenFOAM®. In *Int. Conf. on Multiphase Flow, ICMF*, Jeju, Korea, May 26–31, 2013

T. Marić, H. Marschall, and D. Bothe. lentFoam - A hybrid Level Set/Front Tracking method on unstructured meshes. *Comput. Fluids*, 113:20–31, 2015. ISSN 00457930. doi: 10.1016/j.compfluid.2014.12.019





---

## ACKNOWLEDGMENTS

---

Deep appreciation and heartfelt gratitude I extend to my PhD thesis adviser, Professor Dr. rer. nat. Dieter Bothe for supporting me in the development of both methods. I especially appreciate the discussions about the geometrical Volume-of-Fluid method, that lead to the development of the flux polyhedron triangulation. Thank you Dieter for making that bet with me, I have learned so much from you, not just about science, but also about life.

I am very grateful for the constructive comments given by Prof. dr. sc. Hrvoje Jasak that have significantly helped shape the text of this thesis into its final form.

I have also greatly benefited from discussions with Dr.-Ing Holger Marschall about the problems I have encountered during the method development, especially regarding the volume error redistribution algorithm. His help during the preparation of this thesis is also greatly appreciated.

I have enjoyed the opportunity to concentrate my work on numerical method development and for that I extend my gratitude to the the German Research Foundation (DFG) Cluster of Excellence EXC 259 for funding this research project. The use of the high performance computing resources of the Lichtenberg High Performance Cluster at the TU Darmstadt is also gratefully acknowledged.

To Kerstin Schmitt: thank you Kerstin for reminding me often that there really is such a thing as a completed PhD thesis. Thank you also for keeping the beast of bureaucracy at bay!

Working in the Mathematical Modeling and Analysis research group at TU Darmstadt has taught me a lot. I have enjoyed discussions with other group members, especially about OpenFOAM with the colleagues from the OpenFOAM squad: Daniel Deising, Manuel Falcone, Dirk Gründing, Kathrin Dieter-Kissling, Matthias Niethammer, Chiara Pesci and Paul Weber.

Following the rule of two, this thesis allowed only one master thesis student and now my colleague, Tobias Tolle. Tobi: it has been really fun to develop further the LENT method with you, I am looking forward to the continuation of our joint work!

Final thanks I give to my wife Maria, my parents Iva and Mate, my brother Mario and my friends, who know all too well where I started from, what the path was like, and who supported me in every step of the way.



---

## CONTENTS

---

Nomenclature xxiv

## I Foundations 1

1	INTRODUCTION	3
2	MATHEMATICAL MODEL	7
3	UNSTRUCTURED FINITE VOLUME METHOD	11
3.1	Domain discretization	11
3.1.1	Spatial discretization	11
3.2	Equation discretization	13
3.2.1	Temporal term	16
3.2.2	Convective term	16
3.2.3	Diffusion term	17
3.2.4	Source terms	17
3.2.5	Discretized generalized transport equation	18
3.2.6	Interpolation	18
3.2.7	Boundary conditions	19
3.3	Solution of the linear algebraic equation system	19
3.3.1	Source term linearization	20

## II Geometrical Volume-of-Fluid Method 21

4	METHOD OVERVIEW	23
4.1	Lagrangian / Eulerian flux-based method	24
4.1.1	Discretization of the volume fraction equation	24
4.1.2	Interface reconstruction	29
4.1.3	Volume fraction advection	32
4.2	Lagrangian backward tracing / Eulerian remapping	34
5	LITERATURE SURVEY	37
5.1	Interface reconstruction	37
5.1.1	Interface orientation	39
5.1.2	Interface positioning	48
5.2	Volume fraction advection	51
5.3	Conclusions	70
6	DEVELOPED METHOD	71
6.1	Volume fraction initialization	73

6.2	Interface reconstruction	80
6.2.1	Interface orientation and positioning	80
6.3	Volume fraction advection	85
6.3.1	Point displacement integration	86
6.3.2	Narrow band calculation	88
6.3.3	Flux polyhedron calculation	90
6.3.4	Phase flux volume calculation	103
6.3.5	Parallel implementation	106
6.4	Geometrical operations	107
6.5	Conservative error redistribution	113
6.6	Software design and geometrical transport	118
6.6.1	A C++ generic geometrical library	119
6.6.2	A C++ generic transport library	127
6.6.3	Conclusions	129
6.7	A note on the CFL condition	130

### III Hybrid Level Set / Front Tracking Method 131

7	METHOD OVERVIEW	133
8	LITERATURE SURVEY	137
9	DEVELOPED METHOD	141
9.1	Computing the signed distance fields	142
9.1.1	Unstructured mesh search operations	144
9.1.2	Narrow band generation and propagation	146
9.1.3	Narrow band properties	149
9.2	Reconstructing the front as an iso-surface	149
9.3	Computing the marker field	150
9.4	Evolving the front	151

### IV Results and outlook 155

10	INTRODUCTION	157
11	GEOMETRICAL VOF METHOD	161
11.1	Translation	161
11.2	Rotation	164
11.3	Shear	170
12	LEVEL SET / FRONT TRACKING METHOD	177
12.1	Translation	177
12.2	Rotation	178
12.3	Shear	179
13	SUMMARY AND OUTLOOK	185

13.1 Summary	185
13.2 Outlook	186

## **V Appendix** **187**

A MINIMIZATION OF THE LLSG ERROR	189
----------------------------------	-----

B INTERFACE MESH GENERATION	191
-----------------------------	-----

B.1 Interface surface mesh	191
----------------------------	-----

B.2 Interface volume mesh	191
---------------------------	-----

BIBLIOGRAPHY	195
--------------	-----

Index	209
-------	-----

---

## LIST OF FIGURES

---

Figure 1	Breakup and coalescence of fluid interfaces, photographed by Sackton [117]. 7
Figure 2	Control volume $V$ , its boundary $\partial V$ and the interface modeled with the indicator function $I_i(\mathbf{x}, t)$ . 8
Figure 3	A polyhedral finite volume. 12
Figure 4	Schematic representation of a phase flux volume $V_f^\alpha$ constructed as an intersection between the fluid interface and the flux volume $V_f$ with nonlinear ruled surfaces. 26
Figure 5	A temporally first-order accurate approximation of the phase flux volume. 27
Figure 6	A 2D schematic representation of the phase flux volume $V_f^\alpha$ calculation. The image on the left schematically represents the information required for geometrically approximating the flux volume $V_f$ . The image on the right side shows the phase flux volume contributions $V_{f,c}^\alpha$ computed from the flow configuration shown on the left. 29
Figure 7	$\alpha_c$ field initialization. 31
Figure 8	Reconstructed interface from the field $\alpha_c$ shown in figure 7(d). 32
Figure 9	A complex flux polyhedron with non-planar and non convex ruled surfaces. Using different triangulations for a chosen face (cap polygon $ABCD$ ) of this flux polyhedron result in different flux volume $V_f$ magnitude. 33
Figure 10	A schematic diagram of the discrete Lagrangian tracing / Eulerian remapping (LE) geometrical Volume-of-Fluid (VoF) method. 35
Figure 11	The number of cell faces determines size of $\mathcal{C}$ which in turn determines the accuracy of $\nabla_c$ . Introducing vertex-cell neighbors as proposed by Mavriplis [83] requires explicit stencil reconstruction on hexagonal and hexahedral unstructured meshes, because there exist cells in those meshes that are mutually connected only by their corner points. 41
Figure 12	The reconstructed $\mathbf{x}_c^r$ and advected $\mathbf{x}_c^a$ phase centroid in an interface cell used by the Moment of Fluid (MoF) orientation algorithm. 46
Figure 13	A schematic representation of the interface positioning. The volume fraction function $\alpha_c(\mathbf{p}_c)$ is shown for a fixed interface normal orientation $\mathbf{n}_c$ . Irrespective of $\mathbf{n}_c$ and the shape of the cell, the $\alpha_c$ function has diminishing gradients at the interval endpoints. 48

Figure 14	Volume fraction $\alpha_c$ as a function of the interface plane position $\mathbf{p}_c$ computed using the same $\mathbf{n}_c = (1, 1, 0)$ for three different cell shapes. 49	
Figure 15	A schematic representation of the 2D Rider-Kothe algorithm [111]. 53	
Figure 16	A schematic representation of the 2D stream scheme [53]. 55	
Figure 17	A schematic representation of the two-dimensional DDR scheme [52]. 56	
Figure 18	A schematic representation of the corner-flux ignored by the Defined Donating Region (DDR) scheme. 57	
Figure 19	A schematic representation of the EMFPA scheme. 58	
Figure 20	A schematic representation of the PCFSC advection scheme. 61	
Figure 21	A schematic representation of the initial FMFPA-3D geometrical flux volume. 63	
Figure 22	A schematic representation of the linearized FMFPA-3D geometrical flux volume. 64	
Figure 23	Schematic representation of the volume fraction initialization algorithm of Ahn and Shashkov [7]. The randomly generated points are schematically shown as circles of different colors: those that are inside $\Gamma_h$ are of white color and they contribute to $n_{obj}^p$ in algorithm 2. Obviously in this schematic case, all the points that are inside $\Gamma_h$ are not inside the triangular cell $\mathbf{C}_c$ ; however they are used to compute $\alpha_c$ . 75	
Figure 24	Volume fraction initialization of a verification case from chapter 11 using the Cell / Cell Intersection (CCI) algorithm. The interface $\Gamma_h$ is modeled as an unstructured volume mesh. The exact volume fraction value $\alpha_c$ is computed using $\mathbf{C}_c \cap \Gamma_h$ . left image shows an overlay of $\Gamma_h$ and a cell $\mathbf{C}_c$ , and the right image contains the actual result of a geometrical intersection $\Gamma_h \cap \mathbf{C}_c$ . 77	
Figure 25	The Surface Mesh / Cell Intersection (SMCI) algorithm intersects cell sphere triangles $N_c^{\mathcal{B}_c}$ with the triangulation of a cell $\mathbf{C}_c$ , $T_c$ to set the volume fraction value $\alpha_c$ . The signed distance $\phi_c$ field obtained by solving equation (151) is used to mark the cells that are inside, or outside $\Gamma_h$ . In this case, the narrow band of cells of $\Omega_h$ that surround $\Gamma_h$ is 3 cells wide. 79	
Figure 26	Narrow band flux volumes (gray volumes with black boundaries) and the Piecewise Linear Interface Calculation (PLIC) interface elements (white lines), computed for a circular interface and the 2D shear velocity field from section 11.3, on an unstructured hexahedral mesh. 91	
Figure 27	Different triangulations of a cube. 92	
Figure 28	Different triangulations of a non-convex flux volume. 93	
Figure 29	Different triangulations applied on a non-convex polygon. 95	

- Figure 30 Continuity of the edge triangulation. Edge triangulation results in a unique definition of the triangulation point for two adjacent flux polyhedra, as well as the consistent orientation of the normal area vectors that belong to both triangulations, without requiring any special case handling for non-convexity. 96
- Figure 31 Flux-aware triangulation applied on a non-convex polyhedron with non-convex and non-planar faces. Upper image shows the rendering of the test flux polyhedron in a visualization software that relies on oriented triangulation for visualizing 3D polyhedra. Lower image shows the geometrical result of the flux-aware triangulation applied on the upper polyhedron. Base polygon points, their respective displacements, displaced points and resulting triangulation points follow the notation used in algorithm 12. 98
- Figure 32 Iterative and pyramid correction of for volume conservation. 100
- Figure 33 Problematic thin flux polyhedra generated for the 2D shear verification case by the iterative flux correction for volume conservation. With  $CFL = 1$ , the size of the flux polyhedra becomes comparable to the cell size. 102
- Figure 34 A cell  $\mathbf{C}_c$  with a single intersection half-space  $\mathcal{H}_c^i$  with the flux volume  $V_f$ . 105
- Figure 35 Phase flux volumes computed for the 3D shear verification case of Liovic et al. [73] ( $t = 1.5\text{s}$ ,  $N = 32$ ,  $CFL = 0.5$ ) from two perspectives. 106
- Figure 36 Topologically challenging intersection configurations for a polygon and a half-space resolved by tolerance-based logic. 109
- Figure 37 Parameterized tests for the perturbed intersection between a triangulated star shape and a cube, where the volume is exactly known. 112
- Figure 38 Level Set / Front Tracking (LENT) signed distance  $\phi_c$  calculated in the narrow band for the 2D shear verification case with  $256^2$  cells,  $CFL = 1$ . Interface (surface mesh) is colored white. 133
- Figure 39 LENT marker (volume fraction) field  $\alpha_c$  for the 2D shear case with  $256^2$  cells,  $CFL = 1$ . 135
- Figure 40 Signed distance calculation computed separately for each of the two example points ( $\mathbf{x}_1$ ,  $\mathbf{x}_2$ ) and the front  $\mathcal{F}$ . For the point  $\mathbf{x}_2$ , the shortest ray to  $\mathcal{T}_N$  does not intersect with the triangle, so the signed distance is computed using the closest triangle point  $\mathbf{v}_{\mathcal{T}_N,2}$ . 143



Figure 41	The known-vicinity search algorithm operating on a polygonal mesh in $k$ iterations. The dashed lines represent the topological connections between the current cell and the neighboring cells. Of all the possible connections, only one is followed by the algorithm to the next cell and is visualized as an arrow line. The next cell is the neighbor cell whose center has the minimal distance to the point which is searched for. 147	
Figure 42	The known-vicinity search algorithm operating on a tetrahedral mesh with an obstacle between the seed cell and the point in form of a sphere. Stepping cells ( <i>minCell</i> cells on figure 41) build a search path between the cell and the point. The tetrahedral mesh used for this example is of low quality, which can be seen in the large discrepancy of cell dihedral angles as well as in the sizes of the cells in the search path. 148	
Figure 43	Different approaches to tetrahedral decomposition. 150	
Figure 44	Cylindrical mesh used for two-dimensional verification cases. 161	
Figure 45	<i>Eg</i> geometrical error as a function of $CFL^{-1}$ for the two-dimensional translation case. Results are obtained with the Unsplit Face-Vertex Flux Calculation ( <b>UFVFC</b> )-Youngs combination using the Taylor integration for the cell displacements, Inversed Distance Weighted ( <b>IDW</b> ) point interpolation, Taylor-Backward Differencing Scheme ( <b>TBDS</b> ) flux integration, and pyramid flux volume correction. 162	
Figure 46	<i>Eg</i> geometrical error as a function of $CFL^{-1}$ for the two-dimensional translation case. Results are obtained with the <b>UFVFC</b> -Distance Gradient Normal Reconstruction ( <b>DGNR</b> ) combination with 2 reconstruction steps, using the Taylor integration for the cell displacements, <b>IDW</b> point interpolation, <b>TBDS</b> flux integration, and pyramid flux volume correction. 163	
Figure 47	3D shear verification case of Liovic et al. [73] with $CFL = 0.5$ and $N = 128$ , using the new <b>DGNR</b> reconstruction algorithm with the parameters that correspond to table 17. Polygons that build the <b>PLIC</b> interface are visualized with $\epsilon_r = 1e - 09$ : no artificial interface separation or wisps are visible. 175	
Figure 48	Visual comparison of the temporal integration error of the Euler and Taylor displacement integration rule, isolated for the translation verification case for $N = 32, T = 1, CFL = 3.0$ . 179	
Figure 49	Visualization of the front at the initial, half and final period of motion for the <b>LENT</b> two-dimensional shear case with the Euler explicit temporal integration and the <b>IDW</b> point-velocity interpolation. 182	

Figure 50	Visualization of the front at the initial, half and final period of motion for the <b>LENT</b> two-dimensional shear case with the Taylor explicit temporal integration and the <b>IDW</b> point-velocity interpolation. 183
-----------	--

---

## LIST OF TABLES

---

Table 1	PLIC reconstruction algorithms 38
Table 2	Geometrical algorithms and models. 108
Table 3	Computer architectures used for the verification cases. 158
Table 4	Reference $E_g$ errors of the 2D rotation test done by Liovic et al. [73]. The numbers $N = 32, 64, 128$ represent different mesh resolutions $N^2$ . Numbers inbetween rows define the orders of convergence. 164
Table 5	<i>Reference</i> $E_g$ errors (E) and orders of convergence (O) of the 2D rotation test by López et al. [75], with $CFL \approx 0.5$ corresponding to 201 time steps for the $32 \times 32$ case. 165
Table 6	Results of the 2D rotation test case with the Youngs reconstruction, Taylor integration for the cell displacements, <b>IDW</b> point interpolation, <b>TBDS</b> flux integration, and pyramid flux volume correction. Times are reported for the execution on a single process on the I0 architecture. 165
Table 7	Results of the 2D rotation test case with the Swartz reconstruction [37] with 10 normal correction steps, Taylor integration for the cell displacements, <b>IDW</b> point interpolation, <b>TBDS</b> flux integration, and pyramid flux volume correction. Times are reported for the execution on a single process on the I0 architecture. 166
Table 8	Results of the 2D rotation test case with the Cell Cutting Normal Reconstruction ( <b>CCNR</b> ) reconstruction with 3 reconstruction steps, Taylor integration for the cell displacements, <b>IDW</b> point interpolation, <b>TBDS</b> flux integration, and pyramid flux volume correction. Times are reported for the execution on a single process on the I0 architecture. 166
Table 9	Results of the 2D rotation test case with the <b>DGNR</b> reconstruction with 2 reconstruction steps, Taylor integration for the cell displacements, <b>IDW</b> point interpolation, <b>TBDS</b> flux integration, and pyramid flux volume correction. Times are reported for the execution on a single process on the I0 architecture. 167

Table 10	Algorithm serial execution time and error ratios for the 2D rotation case. Ratios are computed with respect to the Youngs' reconstruction method: Eg S-Y is the ratio of the Eg error between the Swartz and the Youngs reconstruction algorithm. Reconstruction algorithms are used with the same combination of the cell displacement, point-cell interpolation, flux integration and flux correction methods as in tables 6 to 9. 169
Table 11	Results of the 2D rotation test case with the DGNR reconstruction with 2 reconstruction steps, Taylor integration for the cell displacements, IDW point interpolation, TBDS flux integration, and scaled flux volume correction. Times are reported for the execution on a single process on the I0 architecture. 169
Table 12	Results of the 2D shear flow test case on equidistant Cartesian meshes summarized by Comminal et al. [26], presented here for comparison. 171
Table 13	Comparison of the geometrical error of the UFVFC scheme configuration in table 16 with the Cell-wise Conservative Unsplit (CCU) method of Comminal et al. [26] and Owkes-Desjardines (OD) method of Owkes and Desjardins [98]. Values for $T = 0.5, 2$ for the OD method are taken from [98, table 3]. 171
Table 14	Results of the 2D shear test case with the Youngs' reconstruction trapezoidal point displacement integration with the IDW Gauss gradient for the normal orientation, IDW point velocity interpolation, trapezoidal flux integration, and pyramid flux volume correction. Times are reported for the execution on a single process on the I0 architecture. 172
Table 15	Results of the 2D shear test case with the Youngs' reconstruction trapezoidal point displacement integration with the Least Squares (LS) Gauss gradient for the normal orientation, IDW point velocity interpolation, trapezoidal flux integration, and pyramid flux volume correction. Times are reported for the execution on a single process on the I0 architecture. 173
Table 16	Results of the 2D shear test case with the Swartz reconstruction with 10 normal correction iterations, trapezoidal integration for the cell displacements, IDW point interpolation, trapezoidal flux integration, and pyramid flux volume correction. Times are reported for the execution on a single process on the I0 architecture. 174

Table 17	Results of the 3D shear test case with the <b>DG NR</b> reconstruction with 2 normal reconstructions, trapezoidal integration for the cell displacements, <b>IDW</b> point interpolation, trapezoidal flux integration, and pyramid flux volume correction. Times are reported for the execution on a single process on the I1 architecture. 174
Table 18	Results of the <b>LE NT</b> 2D translation test case using the explicit first-order accurate Euler temporal integration and <b>IDW</b> point velocity interpolation. 178
Table 19	Results of the <b>LE NT</b> 2D translation test case using the explicit third-order accurate Taylor temporal integration and <b>IDW</b> point velocity interpolation. 178
Table 20	Results of the <b>LE NT</b> 2D rotation test case using the explicit Taylor temporal integration and <b>IDW</b> point velocity interpolation. 180
Table 21	Results of the <b>LE NT</b> 2D shear test case using the explicit Euler temporal integration and <b>IDW</b> point velocity interpolation. 181
Table 22	Results of the <b>LE NT</b> 2D shear test case using the explicit Taylor temporal integration and <b>IDW</b> point velocity interpolation. 181

---

## LIST OF ALGORITHMS

---

Algorithm 1	Ray crossing point-in-polyhedron by O'Rourke [96]	73
Algorithm 2	Volume fraction initialization : Ahn and Shashkov [7]	74
Algorithm 3	<b>CCI</b> volume fraction initialization	76
Algorithm 4	<b>SMCI</b> volume fraction initialization	78
Algorithm 5	Youngs' / Swartz reconstruction	83
Algorithm 6	Distance Gradient Normal Reconstruction ( <b>DG NR</b> )	85
Algorithm 7	Unsplit Face-Vertex Flux Calculation ( <b>UFVFC</b> )	86
Algorithm 8	Face in narrow-band Face in Narrow Band test ( <b>FNB</b> )	90
Algorithm 9	Oriented convex polygon triangulation Oriented Convex Polygon Triangulation ( <b>OCPT</b> )	92
Algorithm 10	Barycentric convex polygon triangulation Barycentric Convex Polygon Triangulation ( <b>BCPT</b> )	93
Algorithm 11	Edge triangulation	94
Algorithm 12	Flux-aware triangulation	97
Algorithm 13	Phase flux volume calculation	104
Algorithm 14	Polygon $\mathcal{Q}$ / half-space $\mathcal{H}$ intersection	110
Algorithm 15	Polyhedron $\mathcal{S}$ / half-space $\mathcal{H}$ intersection	111

Algorithm 16	Conservative error redistribution	115
Algorithm 17	Process-internal redistribution	115
Algorithm 18	Process-boundary correction	117
Algorithm 19	Volume redistribution	118
Algorithm 20	Level Set / Front Tracking ( <b>LENT</b> )	142
Algorithm 21	Octree-based search	145
Algorithm 22	Known vicinity search	146
Algorithm 23	Point in convex polyhedron	146
Algorithm 24	Naive narrow band propagation	149
Algorithm 25	Mesh-to-front interpolation	152

---

## LISTINGS

---

6.1	Ambiguous volume calculation . . . . .	121
6.2	Tag trait structure. . . . .	122
6.3	value-based tag dispatch. . . . .	122
6.4	Type-based tag dispatch. . . . .	123
6.5	Polygon sequence / half-space distance. . . . .	124
6.6	Indexed polygon / half-space distance. . . . .	124
6.7	Polygon sequence / half-space intersection . . . . .	125
6.8	Swartz interface reconstruction . . . . .	128
6.9	<b>UFVFC</b> advection . . . . .	128
6.10	Phase flux volume calculation . . . . .	129
B.1	Cylinder surface mesh parameter . . . . .	192
B.2	Cylinder volume mesh parameters . . . . .	193

---

## ACRONYMS

---

AABB Axis-Aligned Bounding Box

ALE Arbitrary Lagrangian-Eulerian

AMR Adaptive Mesh Refinement

BDS Backward Differencing Scheme

BE Backward Euler

BGL Boost Geometry Library

BT Barycentric Triangulation

BCPT Barycentric Convex Polygon Triangulation

CAD Computer Aided Design

CCI Cell / Cell Intersection

CCU Cell-wise Conservative Unsplit

CCNR Cell Cutting Normal Reconstruction

CDS Central Differencing Scheme

CG Computer Graphics

CGAL Computational Geometry Algorithms Library

CIAM Calcul d'Interface Affine par Morceaux

CLCIR Conservative Level Contour Interface Reconstruction

CVTNA Centroid Vertex Triangle Normal Averaging

CFD Computational Fluid Dynamics

CFL Courant-Friedrichs-Lewy

CPU Central Processing Unit

CSG Computational Solid Geometry

DG Discontinuous Galerking Method

DR Donating Region

DRACS Donating Region Approximated by Cubic Splines

DDR Defined Donating Region

DGNR Distance Gradient Normal Reconstruction

DNS Direct Numerical Simulations

EGC Exact Geometric Computation

EI-LE Eulerian Implicit - Lagrangian Explicit

EILE-3D Eulerian Implicit - Lagrangian Explicit 3D

EILE-3DS Eulerian Implicit - Lagrangian Explicit 3D Decomposition Simplified

ELVIRA Efficient Least squares Volume of fluid Interface Reconstruction Algorithm

EMFPA Edge-Matched Flux Polygon Advection

EMFPA-SIR Edge-Matched Flux Polygon Advection and Spline Interface Reconstruction

FDM Finite Difference Method

FEM Finite Element Method

FMFPA-3D Face-Matched Flux Polyhedron Advection

FNB Face in Narrow Band test

FV Finite Volume

FVM Finite Volume Method

HPC High Performance Computing

HyLEM Hybrid Lagrangian–Eulerian Method for Multiphase flow

IDW Inversed Distance Weighted

IDWGG Inversed Distance Weighted Gauss Gradient

LENT Level Set / Front Tracking

LE Lagrangian tracing / Eulerian remapping

LEFT hybrid level set / front tracking

LFRM Local Front Reconstruction Method

LVIRA Least squares Volume of fluid Interface Reconstruction Algorithm

LLSG Linear Least Squares Gradient

LS Least Squares

LLSF Linear Least Squares Fit

IDWLSG Inverse Distance Weighted Least Squares Gradient

LCRM Level Contour Reconstruction Method

LSG Least Squares Gradient

MoF Moment of Fluid

MS Mosso-Swartz

NS Navier-Stokes

NP Non-deterministic Polynomial  
 OOD Object Oriented Design  
 OD Owkes-Desjardines  
 OT Oriented Triangulation  
 OCPT Oriented Convex Polygon Triangulation  
 PDE Partial Differential Equation  
 PAM Polygonal Area Mapping Method  
 iPAM improved Polygonal Area Mapping Method  
 PIR Patterned Interface Reconstruction  
 PCFSC Piecewise Constant Flux Surface Calculation  
 PLIC Piecewise Linear Interface Calculation  
 RKA Rider-Kothe Algorithm  
 RK Runge-Kutta  
 RTT Reynolds Transport Theorem  
 SCL Space Conservation Law  
 SMCI Surface Mesh / Cell Intersection  
 SFINAE Substitution Failure Is Not An Error  
 SIR Spline Interface Reconstruction  
 SLIC Simple Line Interface Calculation  
 STL Standard Template Library  
 TBDS Taylor-Backward Differencing Scheme  
 UFVFC Unsplit Face-Vertex Flux Calculation  
 VoF Volume-of-Fluid  
 YSR Youngs' / Swartz Reconstruction algorithm

---

## NOMENCLATURE

---

*Eb* Numerical boundedness error.



$Eg$	Geometrical advection error.
$En$	normalized geometrical (advection) error
$Ev$	Volume conservation error.
$\omega_{r2D}$	Angular velocity of the 2D rotation test.
$\mathbf{x}_p$	Cell centroid.
$N_f$	Cell index of a face-neighbor cell.
$P_f$	Cell index of a face-owner cell.
$\mathbb{C}$	Cell of a mesh $\Omega_h$ .
$\mathbb{C}$	Cell polyhedron.
$r_c$	Cell sphere radius.
$N_c^{\mathcal{B}_c}$	Cell sphere triangles.
$\mathcal{B}_c$	Cell sphere.
$\mathbf{c}_{r2D}$	Center of the circle in the 2D rotation test.
$\mathbf{c}_{s2D}$	Center of the circle in the 2D shear test.
$\mathbf{c}_{s3D}$	Center of the circle in the 2D shear test.
$\Omega_h$	Discretized solution domain.
$\mathbf{x}_f$	Face centroid.
$\mathbb{F}$	Face of a cell $\mathbb{C}$ .
$\mathbf{S}_f$	Face surface area normal vector.
$N_{f,c}$	Face-point adjacent faces.
$N_{F_f}$	Flux stencil.
$\mathcal{F}$	Front mesh.
$\mathcal{H}$	Halfspace.
$N_{c,\alpha_c}$	Indices of interface cells.
$N_c^i$	Interface cells.
$\mathcal{Q}_c$	Interface polygon.
$\mathcal{H}_c^i$	Intersection half-space.
$\mathbf{n}_{f',l}$	Linearized swept face normal.

$N_c^p$	Number of cells whose corner point is the point $p$ .
$N_{f,f}$	Number of edge-adjacent faces of face $f$ .
$N_{c,c}$	Number of face-adjacent cells of cell $c$ .
$S$	Parallel speedup.
$\Lambda_f$	Phase flux volume scaling coefficient.
$N_{pc}$	Point $\rightarrow$ cell labels.
$I_p$	Point addressing function.
$w_{pc}$	Point-cell inverse distance weight.
$R_{r2D}$	Radius of the circle in the 2D rotation test.
$R_{s2D}$	Radius of the circle in the 2D shear test.
$R_{s3D}$	Radius of the circle in the 2D shear test.
$a_{r2D}$	Rotation axis of the 2D rotation test.
$\mathbb{P}$	Set of mesh points.
$\Pi$	Set permutation.
$\lambda$	Signed distance diffusion coefficient.
$T_{r2D}$	Simulation time of the 2D rotation test.
$T_{s2D}$	Simulation time of the 2D shear test.
$T_{s3D}$	Simulation time of the 2D shear test.
$D$	Spatial dimension.
$\psi$	Squared search distance.
$\delta_t^{tr} \mathbf{x}$	Taylor series point displacement.
$\delta_t^{tr} \mathbf{x}$	Trapezoidal point displacement.
$\mathcal{M}_c(\mathcal{T})$	Triangle $\rightarrow$ cell map.
$\mathcal{N}(\mathcal{T}_k)$	Triangle neighborhood.
$\mathbf{n}_{\mathcal{T}}$	Triangle normal area vector.
$\mathcal{T}_k$	Triangle of the front mesh.
$\mathbf{v}_{\mathcal{T},i}$	Vertex that belongs to a triangle $i$ of the front mesh.
$\alpha_p$	Volume fraction stored at a mesh point (cell corner point).
$\alpha_f$	Volume fraction stored at the face center.

# **Part I**

## **Foundations**



---

## INTRODUCTION

---

Numerical methods for Direct Numerical Simulations (**DNS**) of two-phase flows are actively being investigated and a multitude of new methods have recently been developed in order to improve accuracy, reduce computational costs and extend methods with the goal of predictively simulating more complex physical processes. At the core of each two-phase **DNS** lies a numerical method that approximates the evolution of a fluid interface. When the phases do not mix, a sharp interface forms that separates them. The evolution of the sharp interface is difficult to handle numerically, because the interface represents an evolving discontinuity that along with a strong deformation experiences topological changes such as breakup and coalescence. The majority of developed numerical methods for two-phase **DNS** that are of high fidelity are developed in two dimensions and on structured meshes, as the method complexity becomes substantially higher in  $3D$  and on unstructured meshes. It is difficult to maintain volume (mass) conservation, numerical boundedness and a stable order of convergence, without introducing complex geometrical operations or conservative error redistribution. An important task of any method is the ability to simulate problems of technical complexity, that involve handling solution domains that are geometrically complex. The sizes of both technical problems and those of pure academic interest require numerical methods that support not only efficient parallel execution on modern computer architectures, but also serial efficiency that ensures reasonable execution times. The requirements of any two-phase **DNS** that can handle geometrically complex solution domains have been concisely summarized in the following way: “We seek an algorithm that:

- is globally and locally mass conservative;
- maintains at least second-order accuracy in time and space;
- maintains compact interface discontinuity width;
- is topologically robust;
- is amenable to three-dimensions;
- is amenable to general unstructured meshes;
- can accomodate additional interfacial physics models;
- can track interfaces bounding more than two materials;
- is computationally efficient;

- can be implemented by novices; and
- can be readily maintained, improved, and extended.

” (verbatim quote from Kothe [70].) In the best case scenario, a new numerical method will result in advances in all aforementioned aspects.

Recent advancements done by Shin et al. [129], Le Chenadec and Pitsch [71], Jemison et al. [64], among many others, emphasize combining algorithms of existing methods into new hybrid methods. In a hybrid approach, the modularity and extensibility of the method implementation represent decisive factors in obtaining new insights. More often than not, the method implementation must rely on external software frameworks and libraries for specific complex computations. Code re-use is what has driven the evolution of the software development field from procedural, to structured, object oriented, generic as well as functional programming. The ability to re-use and combine well tested components results in faster method development. This practice from the field of Software Development can and should be transferred to the development of two-phase **DNSs**, because their complexity causes uncertainty that can only be handled by flexibility in exchanging and extending method sub-algorithms. Algorithms that are orthogonal in the software development sense are independent of each other and can be extended and interchanged without incurring modifications of the numerical method. Proper use of orthogonal algorithms represent the basis for building two-phase **DNSs**, regardless of the used programming language.

One aim of this thesis is to demonstrate that significant improvements to the simulation of two-phase flows on unstructured meshes can be achieved by following modern software development practices. This thesis comprises of an investigation of modern methods used for two-phase **DNS** and the development of two novel methods using the unstructured Finite Volume Method (**FVM**): the geometrical **VoF** method and the hybrid Level Set / Front Tracking method. Both methods have been implemented as extensions of the OpenFOAM framework for Computational Fluid Dynamics (**CFD**) (Weller, Tabor, Jasak, and Fureby [147], Jasak and Jemcov [62]) and programmed in the C++ programming language.

Although both methods have been implemented within a single software platform, their implementations vary significantly. The geometrical **VoF** method is mainly based on generic programming, because of the need to optimize and freely interchange low-level geometrical sub-algorithms. The hybrid Level Set / Front Tracking method relies on Object Oriented Design (**OOD**), as the algorithms that need to be freely exchanged or extended operate on the level of the whole solution domain. Because there are some similarities between both numerical methods, part **I** of this thesis covers the topics that are in common to both methods. The mathematical model is covered by chapter 2 and the unstructured Finite Volume (**FV**) method implemented in OpenFOAM is briefly described in chapter 3.

The geometrical **VoF** method uses an implicit description of the fluid interface in the form of a fill level of the control volume, which is then used to reconstruct the geometrical representation of the fluid interface. The velocity field is then used together with the reconstructed geometrical data to modify the fill levels in the next time step. Part **II** covers the geometrical **VoF** method in detail. An introduction to

the geometrical **VoF** method is provided in chapter 4, a comparison of the state of the art geometrical **VoF** methods is covered in chapter 5 and the details of both the underlying geometrical operations as well as transport algorithms and their respective software design are given in chapter 6.

The hybrid Level Set / Front Tracking method relies on a mesh of mutually connected triangles to represent the fluid interface, while the topological changes of the interface are based on reconstructing the triangle mesh from a locally computed signed distance field, as a zero level set. The Level Set / Front Tracking (**LENT**) is therefore a hybrid method between the Level Set and Front Tracking methods. Part III contains the chapters that describe the hybrid Level Set / Front Tracking method in detail. Chapter 7 provides a method overview and chapter 8 covers the state of the art methods. In chapter 9, the mesh search operations that are required by the hybrid Level Set / Front Tracking method in order to support unstructured meshes are presented, as well as other important aspects of the method. Since the software design of the **LENT** is significantly simpler than the design of the geometrical **VoF** method, as it is based on standard Object Oriented Design (**OOD**) patterns, its description has been omitted for the sake of brevity.

Different verification tests have been developed by the research community for testing the accuracy and efficiency of developed methods. The test cases as well as the results obtained for both methods are reported in part IV, where the conclusions are drawn on the methods investigated in this work, as well as an outlook towards future developments.

The contributions to both **LENT** and geometrical **VoF** method in this thesis can be summarized by reflecting on the aforementioned list of method properties proposed by Kothe [70]. The proposed geometrical **VoF** method is mass (volume) conservative near machine tolerance and second-order accurate in time and space. The volume conservation error of the **LENT** is second-order convergent and further straightforward improvements are proposed based on the initialization algorithms implemented for the geometrical **VoF** method. Both **LENT** and geometrical **VoF** method can robustly handle topological changes of the interface, and their implementations are applicable to pseudo  $2D^1$  and  $3D$  calculations. The interface discontinuity is sharply resolved by both methods: both methods are void of numerical diffusion, so no artificial separation of the interface occurs. Extensions involving additional physical models are straightforward for the **LENT**, whereas more effort is required for the geometrical **VoF** method. Because of their modular implementation, both methods model the fluid interface and the related method algorithms on a very high level of abstraction. This makes it possible to handle multi-material flows on the level of an end (solver) application, by initializing multiple different interfaces that are evolved using the same abstracted evolution algorithms that belong to the respective method. Of course, handling of the interface reconstruction for multi-material cells would still need to be implemented to accurately handle multi-material flows. Results show that the proposed geometrical **VoF** method has comparable execution times to similar methods on structured Cartesian meshes, even though substantial additional calculations are necessary on

---

<sup>1</sup> A two-dimensional calculation implemented using a single  $3D$  layer of mesh cells.

unstructured meshes. Even though the **LENT** relies on complex search algorithms for the execution on unstructured meshes, avoiding the solution of an additional Poisson equation for the marker field, characteristic for Front Tracking, increases its computational efficiency significantly. The implementation of the geometrical **VoF** method is far from simple, however the proposed software design is based on C++ template metaprogramming for the geometrical algorithms and C++ template policies for the geometrical transport and this simplifies the method extension significantly, by allowing the researcher to concentrate her/his effort on a single geometrical or transport sub-algorithm. The object oriented implementation of the **LENT** allows for straightforward method extension and modification.



---

MATHEMATICAL MODEL

---

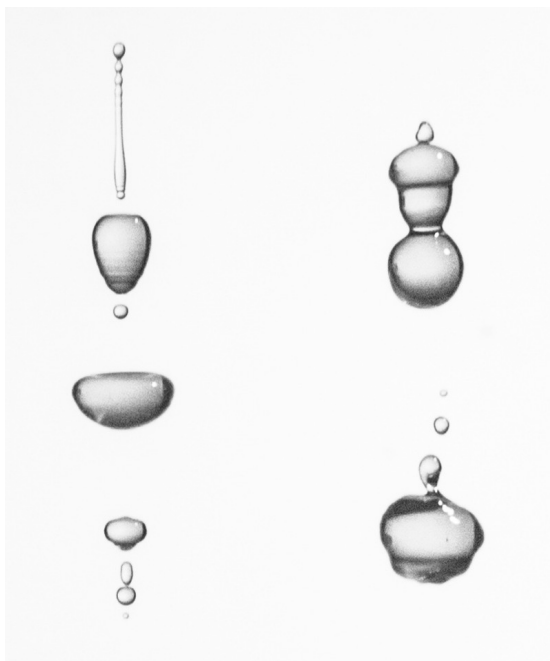


Figure 1: Breakup and coalescence of fluid interfaces, photographed by Sackton [117].

Between immiscible fluid phases, a sharp (discontinuous) interface forms. An example of such a fluid interface between water and air is shown for a set of droplets in the photograph taken by Sackton [117] in figure 1. As the result of volumetric and surface forces acting on the fluid interfaces, both in nature and in technical processes, fluid interfaces experience complex deformation as well as topological changes such as interface separation (breakup) and coalescence. Mathematical modeling of the complex evolution of fluid interfaces represents the crucial foundation for predictive numerical simulations of multiphase flows. Numerical simulations are necessary to solve multiphase flow problems, because exact solutions cannot be obtained for problems that involve complex interface evolution. Nowadays, simulations of multiphase flows bring additional insights into natural and technical processes. They are beginning to play a more important role in the design of technical processes, because they can deliver very detailed insights into processes that are difficult to investigate experimentally, due to either the complexity of the involved physical process, or its scale.

The goal of a mathematical model for the interface evolution is a complete spatial configuration of all the interfaces in a multiphase system at any point in

time. From the known spatial configuration of the fluid interface, it is then possible to model other physical processes occurring in the multiphase system, such as the exchange of momentum, energy and mass on and across fluid interfaces. Numerical methods proposed in this thesis deal only with the motion of fluid interfaces, not the forces that cause them: flow velocity is prescribed.

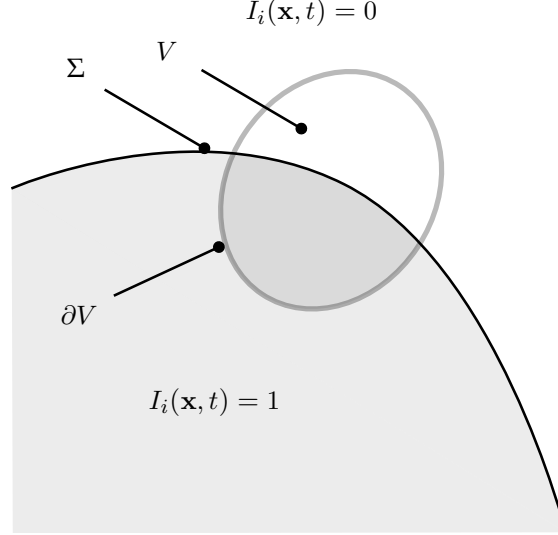


Figure 2: Control volume  $V$ , its boundary  $\partial V$  and the interface modeled with the indicator function  $I_i(\mathbf{x}, t)$ .

A mathematical model for the evolution of the interface  $\Sigma$  is based on the physical law of the conservation of mass, applied to a control volume  $V$  fixed in space, with a boundary  $\partial V$ , as shown in figure 2. Exchanges between the fluid phases that include mass and energy are not taken into account: the phases interact with each other by mechanical forces. In order to distinguish  $n$  immiscible phases, an indicator function is introduced for each phase  $i$  as

$$I_i(\mathbf{x}, t) = \begin{cases} 1 & \text{if phase } i \text{ occupies point } \mathbf{x} \in \mathcal{R}^D \text{ at the time } t, \\ 0 & \text{otherwise,} \end{cases} \quad (1)$$

where  $D$  is the spatial dimension. If  $I_i(\mathbf{x}, t)$  is known for every  $i$ -th phase, then the spatial configuration of all fluid interfaces is fully defined. Therefore, the problem of tracking the interfaces between immiscible fluids is reformulated into a problem of determining  $I_i(\mathbf{x}, t)$ . To determine  $I_i(\mathbf{x}, t)$ , physical mass conservation law can be applied to a fixed *control volume*  $V$ . In order to compute the total mass of all phases contained within  $V$ , density fields of all the fluid phases must be defined. If the phases are incompressible, the density field of each of the phases can be defined using the indicator function

$$\rho_i(\mathbf{x}, t) = I_i(\mathbf{x}, t)\rho_{i,c}, \quad (2)$$

where  $\rho_{i,c}$  is the constant density of the  $i$ -th phase. The mass conservation law of a fixed control volume without the exchange of mass and energy across fluid

interfaces states that the rate of change of the mass given by a phase  $i$  in a volume  $V$  fixed in time is equal to the total mass flow rate of phase  $i$  through its boundaries following the Reynolds Transport Theorem (RTT)

$$\frac{d}{dt} \int_V \rho_i dx = - \int_{\partial V} \mathbf{f}_{m,i} \cdot \mathbf{n} do, \quad (3)$$

where  $\mathbf{f}_{m,i}$  is the mass flux vector of the  $i$ -th phase across the boundary  $\partial V$  of the volume  $V$ . It can be easily shown that

$$\mathbf{f}_{m,i} = \rho_i(\mathbf{x}, t) \mathbf{u}(\mathbf{x}, t) \quad (4)$$

where  $\mathbf{u}(\mathbf{x}, t)$  is the flow velocity. Equations (2) and (4) inserted into equation (3), result in

$$\frac{d}{dt} \int_V I_i(\mathbf{x}, t) dx = - \int_{\partial V} I_i(\mathbf{x}, t) \mathbf{u}(\mathbf{x}, t) \cdot \mathbf{n} do, \quad (5)$$

the integral form of the indicator function evolution equation for the  $i$ -th phase. The entire part II of this thesis is dedicated solely to the approximation of the solution of equation (5). As the preparation step for the approximative solution, the volume fraction of the  $i$ -th phase is introduced as

$$\alpha_{i,V} = \frac{1}{|V|} \int_V I_i(\mathbf{x}, t) dx. \quad (6)$$

Dividing equation (5) by  $V$  and considering equation (6), leads to

$$\frac{d}{dt} \frac{1}{|V|} \int_V I_i(\mathbf{x}, t) dx = \frac{d}{dt} \alpha_{i,V} = - \frac{1}{|V|} \int_{\partial V} I_i(\mathbf{x}, t) \mathbf{u}(\mathbf{x}, t) \cdot \mathbf{n} do. \quad (7)$$

Solving equation (7) starts by integrating the equation in time over an interval  $\delta\tau$

$$\int_{\tau}^{\tau+\delta\tau} \frac{d}{dt} \alpha_{i,V} dt = - \frac{1}{|V|} \int_{\tau}^{\tau+\delta\tau} \int_{\partial V} I_i(\mathbf{x}, t) \mathbf{u}(\mathbf{x}, t) \cdot \mathbf{n} do dt, \quad (8)$$

which results in the exact integral form of the volume fraction equation

$$\alpha_{i,V}(\tau + \delta\tau) = \alpha_{i,V}(\tau) - \frac{1}{|V|} \int_{\tau}^{\tau+\delta\tau} \int_{\partial V} I_i(\mathbf{x}, t) \mathbf{u}(\mathbf{x}, t) \cdot \mathbf{n} do dt. \quad (9)$$

Equation (9) is still an exact equation, as no approximations have been used whatsoever. It is important to notice that equation (9) has two variables,  $\alpha_{i,V}$  and  $I_i(\mathbf{x}, t)$ . In order for this equation to be solvable, two exact maps must exist  $\mathcal{M}_{I,\alpha} : \alpha_{i,V} \rightarrow I_i(\mathbf{x}, t)$  within  $V$ , and  $\mathcal{M}_{\alpha,I} : I_i(\mathbf{x}, t) \rightarrow \alpha_{i,V}$  within  $V$ . The difficulty in solving equation (9) exactly is the unavailability of exact maps  $\mathcal{M}_{I,\alpha}, \mathcal{M}_{\alpha,I}$  for general configurations of  $(I_i(\mathbf{x}, t), V)$ , as well as the impossibility to compute the r.h.s term of equation (9) exactly for general  $I_i(\mathbf{x}, t)$  and  $\mathbf{u}(\mathbf{x}, t)$ .

The discontinuity of the fluid interface poses a substantial problem in approximating the solution of equation (9). Numerical methods based on interpolation

have large errors in the vicinity of discontinuities. An alternative approach relies on the field of Computational Geometry to approximate the maps  $\mathcal{M}_{I,\alpha}, \mathcal{M}_{\alpha,I}$  and the r.h.s of equation (9). Enhancements of such geometrical methods represent one focus of this thesis.

The geometrical **VoF** method presented in part II approximates the solution of equation (9) by constructing approximate counterparts of the exact maps  $\mathcal{M}_{I,\alpha}, \mathcal{M}_{\alpha,I}$  and an approximate computation of the r.h.s. term of equation (9).

The hybrid Level Set / Front Tracking method presented in part III does not approximate the interface evolution by solving equation (9) at all. The **LENT** method approximates  $\mathcal{M}_{I,\alpha}, \mathcal{M}_{\alpha,I}$ , while the evolution of the fluid interface is approximated directly by displacing the geometrical approximation of the fluid interface over time interval  $\delta t$  in a Lagrangian way through the solution domain.

Both the **LENT** method and the geometrical **VoF** method rely on discrete approximations of linear differential operators. Different numerical methods have been developed for the purpose of approximating differential operators, to name some of them: Finite Difference Method (**FDM**), Finite Volume Method (**FVM**), Finite Element Method (**FEM**) and Discontinuous Galerking Method (**DG**). Proposed Lagrangian / Eulerian methods are independent of the choice of the numerical method chosen for the approximation of differential operators. Still, methods presented in this thesis, as well as the vast majority of geometrical methods for two-phase flows, rely on the Finite Volume Method (**FVM**) that has a formal second-order of accuracy. Therefore, a second-order unstructured Finite Volume Method (**FVM**) is chosen for the methods proposed in this thesis and it is covered briefly in the next chapter.

---

## UNSTRUCTURED FINITE VOLUME METHOD

---

As both proposed methods in this thesis are developed with the end goal of simulating technical problems with solution domains of high geometrical complexity, they rely on an unstructured Finite Volume Method (**FVM**) for domain and operator discretization. Specifically, both methods were implemented within the OpenFOAM software platform for **CFD**, so this chapter covers the basis of this version of the **FVM**.

The **FVM** used in OpenFOAM (Weller, Tabor, Jasak, and Fureby [147], Jasak and Jemcov [62]) has been extensively covered in previous works. A large amount of information on this topic can be found in the publicly available PhD theses [61, 68, 141, 82], among others. Additionally, detailed descriptions of the unstructured **FVM** are available in [55, 145]. In order to avoid repetition and keep this text self-sustained, only a brief overview of the **FVM** method on unstructured meshes is provided in this chapter.

Approximative solution procedure of the mathematical model using the unstructured **FVM** is built upon three sequential steps:

1. domain discretization,
2. equation discretization,
3. solving the (linear) algebraic equation system.

### 3.1 DOMAIN DISCRETIZATION

#### 3.1.1 *Spatial discretization*

The continuous solution domain  $\Omega$  is approximated by a discrete solution domain  $\Omega_h$  defined as a union of non-overlapping convex polyhedra (cells)  $C$

$$\Omega_h = \cup_i C_i. \quad (10)$$

$\Omega_h$  is known as *computational mesh* or simply *mesh* and  $C_i$  are called *mesh cells* or simply *cells*. Cells are synonyms for *finite volumes* when the **FVM** is used for equation discretization.<sup>1</sup> A finite volume is shown on figure 3, together with the geometrical information that the **FVM** requires for equation discretization.

---

<sup>1</sup> In this text, the terms *mesh cell*, *cell* and *finite volume* are freely interchanged.

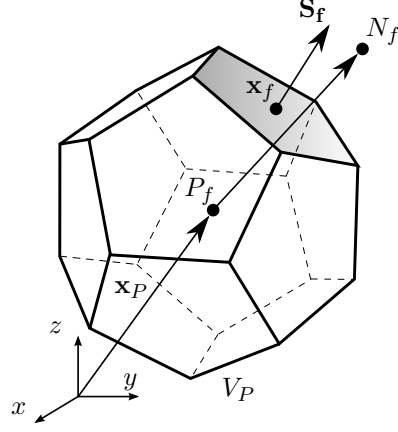


Figure 3: A polyhedral finite volume.

The unstructured **FVM** relies on indirect addressing of mesh data. This reduces the storage requirements for mesh data by preventing information repetition and still maintains  $\mathcal{O}(1)$  complexity when accessing mesh elements. Indirect addressing uses the mesh points as the basic source of all geometrical mesh information. Additional to mesh points, the mesh is defined by topologically higher-level mesh elements, such as: faces, cells, mesh boundary, mesh boundary patches, mesh edges, etc. Those topological sets are all defined as sets of indices that index a lower level set and ultimately refer to mesh points. Topological sets that define the mesh are either *ordered* or *unordered* index sets. The set of all mesh points are defined as

$$\mathbb{P} = \{\mathbf{x}_i \in \mathbb{R}^3 : i = 1, 2, 3, \dots, N_p\}. \quad (11)$$

It is helpful to introduce an index set of the mesh points  $\mathbb{P}$  as

$$L_p = \{1, 2, 3, \dots, N_p\}. \quad (12)$$

A finite volume face  $\mathbb{F}$  of size  $N_f$  is an oriented  $N_f$ -sided polygon defined as an ordered list (a tuple) of indices that uniquely identify some points in  $\mathbb{P}$ :

$$\mathbb{F} = (j_k \in L_p : k = 1, 2, 3, \dots, N_f). \quad (13)$$

The indirect addressing that relates an index  $j_k \in \mathbb{F}$  to a point  $\mathbf{x}_{j_k}$  in  $\mathbb{P}$  can now be defined as a mapping

$$I_{p,f} : \{1, 2, 3, \dots, N_f\} \rightarrow \mathbb{P}, \quad (14)$$

such that

$$\mathbf{x}_{j_k} = I_{p,f}(k). \quad (15)$$

A circular shift of an index  $j$  that belongs to an index set of size  $n$  is defined as

$$\Pi_c(j) \equiv (j \bmod n) + 1, \quad j = 1, \dots, n. \quad (16)$$

A face  $\mathbb{F}$  is invariant to a circular shift permutation because its ordering remains preserved

$$\Pi_c(\mathbb{F}) \equiv \mathbb{F} \equiv \{p_i | i = \Pi_c(j), j = 1, \dots, |\mathbb{F}|\}. \quad (17)$$

A mesh cell  $\mathbb{C}$  of size  $M$  is defined as a set of indices to mesh faces

$$\mathbb{C} = \{f_1, f_2, \dots, f_j, \dots, f_M\}. \quad (18)$$

A cell is therefore invariant to permutations, so

$$\Pi(\mathbb{C}) \equiv \mathbb{C}. \quad (19)$$

For each face  $\mathbb{F}_f$  a face centroid is defined as

$$\mathbf{x}_f = \frac{1}{|\mathbb{F}_f|} \sum_{j=1}^{|\mathbb{F}_f|} I_{p,f}(j). \quad (20)$$

For each cell  $\mathbb{C}$ , a cell centroid can be defined as

$$\mathbf{x}_p = \frac{1}{|\mathbb{C}|} \sum_{i=1}^{|\mathbb{C}|} \frac{1}{|\mathbb{F}_i|} \sum_{j=1}^{|\mathbb{F}_i|} I_{p,f}(j). \quad (21)$$

A surface area normal vector of the face  $\mathbb{F}_f$  can be defined as

$$\mathbf{S}_f = 0.5 \sum_{j=1}^{|\mathbb{F}_f|} (I_{p,f}(j) - \mathbf{x}_f) \times (I_{p,f}(\Pi_c(j)) - \mathbf{x}_f). \quad (22)$$

The orientation of  $\mathbf{S}_f$  will therefore depend on the ordering of  $\mathbb{F}_f$ . As shown in figure 3, for each face  $f$ , the adjacent cells will have two corresponding indexes in the set of mesh cells  $\Omega_h$ ,  $P_f$  and  $N_f$ . The surface area normal vector defined by equation (22) will be oriented from  $P_f$  to  $N_f$ , as shown in figure 3, only if  $P_f < N_f$ . This requirement on the orientation of  $\mathbf{S}_f$  is imposed in order to reduce overall computational complexity of the equation discretization covered in the next section.

### 3.2 EQUATION DISCRETIZATION

Equations whose derivation is based on physical conservation laws all have a similar form. Derivation of governing equations in fluid dynamics and their representation with a general transport equation can be found in CFD books [41, 55, 145]. A scalar transport equation can be used to describe the equation discretization using FVM

$$\partial_t(\phi) + \nabla \cdot (\mathbf{u}\phi) - \nabla \cdot (\gamma_\phi \nabla \phi) = S_\phi, \quad (23)$$

where  $\phi$  is the dependent variable and  $\mathbf{u}$  is a prescribed velocity field. A more general form of equation (23) can be found in [61, 68, 141, 82].

Equation (23), integrated over a time interval  $\delta\tau$  and a control volume  $V$  obtains the form

$$\int_{\tau}^{\tau+\delta\tau} \int_V [\partial_t(\phi) + \nabla \cdot (\mathbf{u}\phi) - \nabla \cdot (\gamma \nabla \phi)] dV dt = \int_{\tau}^{\tau+\delta\tau} \int_V S_{\phi} dV dt. \quad (24)$$

Many temporal discretization schemes as well as interpolation schemes can be used with the unstructured FVM. In this text, only the simplest schemes are used in order to simplify the exposition. More detailed information on equation discretization for the unstructured FVM as well as the derivation of different interpolation schemes is available in [61, 68, 145, 55].

A volume averaged quantity replaces the continuous distribution of  $\phi$  in the cell  $P$  by a piecewise constant cell-wise approximation. The piecewise constant cell-wise approximation is second-order accurate when the value is associated with the center of the finite volume. This can be shown if  $\phi$  is developed in Taylor series around the centroid  $\mathbf{x}_c$

$$\begin{aligned} \int_{V_c} \phi(x) dx &= \int_{V_c} \phi_c + \nabla \phi|_c (\mathbf{x} - \mathbf{x}_c) + \nabla \nabla \phi|_c :: (\mathbf{x} - \mathbf{x}_c)^2 + \dots dx \\ &= \phi_c |V_c| + \nabla \phi|_c \int_{V_c} (\mathbf{x} - \mathbf{x}_c) dx + \nabla \nabla \phi|_c :: \int_{V_c} (\mathbf{x} - \mathbf{x}_c)^2 dx + \dots \\ &= \phi_c |V_c| + \mathcal{O}((\mathbf{x} - \mathbf{x}_c)^2) \end{aligned} \quad (25)$$

where  $(\mathbf{x} - \mathbf{x}_c)^2$  is a tensor product and the definition of the geometrical centroid can be reformulated as

$$\mathbf{x}_c = \frac{\int_{V_c} \mathbf{x} dx}{\int_{V_c} dx} \Rightarrow \int_{V_c} (\mathbf{x} - \mathbf{x}_c) dx = 0. \quad (26)$$

A surface averaged quantity associated with the face center  $\mathbf{x}_f$  is also second-order accurate, since it is a result of reducing the dimensions of equations (25) and (26):

$$\int_{\mathbf{S}_f} \phi(x) \cdot \mathbf{n} do = \phi_f \cdot \mathbf{S}_f + \mathcal{O}((\mathbf{x} - \mathbf{x}_f)^2). \quad (27)$$

Time is discretized into successive intervals called *time steps* ( $\delta\tau$ ). For brevity, the index  $n$  is introduced

$$\phi(\tau + m\delta\tau) \equiv \phi^{n+m} : m = 0, 1, 2, \dots \quad (28)$$

Integration in time is approximated using the trapezoidal rule

$$\int_{\tau}^{\tau+\delta\tau} \phi(t) dt = 0.5 \cdot (\phi^n + \phi^{n+1}) \delta\tau + \epsilon_{tr} \quad (29)$$

where (Atkinson [13, equation 5.1.4])

$$\epsilon_{tr} = -\frac{\delta\tau^3}{12} \phi''(\eta) = \mathcal{O}(\delta\tau^3), \eta \in [\tau, \tau + \delta\tau]. \quad (30)$$



Integration over a control volume  $V_c$  is approximated using the Gauss divergence theorem that states

$$\int_V \nabla \cdot \mathbf{a} dx = \oint_{\partial V} \mathbf{a} \cdot \mathbf{n} do. \quad (31)$$

Here  $\mathbf{a}$  is a vector field, and  $\partial V$  is the boundary surface of the volume  $V$ . From this follows the definition of divergence

$$\nabla \cdot \mathbf{a} = \lim_{V \rightarrow 0} \frac{1}{|V|} \oint_{\partial V} \mathbf{a} \cdot \mathbf{n} ds. \quad (32)$$

If a cubic volume  $V = dx_1 dx_2 dx_3$  with outward facing normal vectors and a center  $\mathbf{x} = (x_1, x_2, x_3)$  in a Cartesian coordinate system with the basis  $(\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3)$  is chosen, it follows that

$$\begin{aligned} \lim_{V \rightarrow 0} \oint_{\partial V} \mathbf{a} \cdot \mathbf{n} ds &= \lim_{V \rightarrow 0} \sum_{i=1, i \neq j \neq k}^3 \left[ \mathbf{a}(x_i + \frac{dx_i}{2}) - \mathbf{a}(x_i - \frac{dx_i}{2}) \right] \mathbf{e}_i dx_j dx_k = \\ &= \lim_{V \rightarrow 0} \sum_{i=1, i \neq j \neq k}^3 \left[ a_i(x_i + \frac{dx_i}{2}) - a_i(x_i - \frac{dx_i}{2}) \right] dx_j dx_k. \end{aligned} \quad (33)$$

If  $\mathbf{a}$  is an analytical function a Taylor series can be developed around  $\mathbf{x}$ :

$$a_i(x_i + \frac{dx_i}{2}) = a_i(x_i) + \frac{da_i}{dx_i} \frac{dx_i}{2} + \frac{d^2 a_i}{dx_i^2} \frac{dx_i^2}{4} + \frac{d^3 a_i}{dx_i^3} \frac{dx_i^3}{12} + O(\frac{dx_i^4}{2 \cdot 4!}), \quad (34)$$

$$a_i(x_i - \frac{dx_i}{2}) = a_i(x_i) - \frac{da_i}{dx_i} \frac{dx_i}{2} + \frac{d^2 a_i}{dx_i^2} \frac{dx_i^2}{4} - \frac{d^3 a_i}{dx_i^3} \frac{dx_i^3}{12} + O(\frac{dx_i^4}{2 \cdot 4!}). \quad (35)$$

Taking the difference of equations (34) and (35) and inserting the result into equation (33), results in

$$\begin{aligned} \lim_{V \rightarrow 0} \oint_{\partial V} \mathbf{a} \cdot \mathbf{n} ds &\stackrel{\lim_{V \rightarrow 0}}{=} \sum_{i=1, i \neq j \neq k}^3 \frac{da_i}{dx_i} dx_i dx_j dx_k + \\ &+ \sum_{i=1, i \neq j \neq k}^3 \frac{d^3 a_i}{dx_i^3} \frac{dx_i^2}{6} dx_i dx_j dx_k + \\ &+ \sum_{l=2}^{\infty} \sum_{(i=1, i \neq j \neq k)}^3 \frac{d^{2l+1} a_i}{dx_i^{2l+1}} \frac{dx_i^{2l+1}}{l!} dx_j dx_k. \end{aligned} \quad (36)$$

Since  $dx_i dx_j dx_k = |dV|$ , and with  $\mathbf{S}_f$  denoting the area normal vector of the face  $f$  of  $V$ , it follows that

$$\begin{aligned} \lim_{V \rightarrow 0} \frac{1}{|V|} \oint_{\partial V} \mathbf{a} \cdot \mathbf{n} ds &= \lim_{V \rightarrow 0} \frac{1}{|V|} \sum_f \int_{\mathbf{S}_f} \mathbf{a} \cdot \mathbf{n} ds \\ &= \nabla \cdot \mathbf{a} + \sum_{i=1}^3 \frac{d^3 a_i}{dx_i^3} \frac{dx_i^2}{6} + \sum_{l=2}^{\infty} \sum_{(i=1, i \neq j \neq k)}^3 \frac{d^{2l+1} a_i}{dx_i^{2l+1}} \frac{dx_i^{2l}}{l!}. \end{aligned} \quad (37)$$

With the limit applied, the second and third terms on the r.h.s. tend faster to zero than the first one, and the definition of the divergence is obtained. However, if  $V$  doesn't tend to zero, but to a finite small number (control or finite volume  $V_c$ ), the first element of the truncation error series is proportional to  $dx_i^2$  which shows the approximation to be second-order accurate. Consequently, the approximation of the Gauss divergence theorem takes on the following form

$$\nabla \cdot \mathbf{a} = \frac{1}{|V_c|} \sum_f \int_{\mathbf{S}_f} \mathbf{a} \cdot \mathbf{n} ds + \mathcal{O}(\|\mathbf{d}\|^2) = \frac{1}{|V_c|} \sum_f \mathbf{a}_f \cdot \mathbf{S}_f + \mathcal{O}(\|\mathbf{d}\|^2) \quad (38)$$

where  $\mathbf{d}$  is the distance between the centers of adjacent cells.

### 3.2.1 Temporal term

The temporal term discretization follows from its integration in space and time

$$\int_{\tau}^{\tau+\delta\tau} \int_{V_c} \partial_t \phi \, dV \, dt = |V_c| \int_{\tau}^{\tau+\delta\tau} (\partial_t \phi)_c \, dt = |V_c| [(\phi)_c^{n+1} - (\phi)_c^n] + \mathcal{O}(\|\mathbf{d}\|^2) \quad (39)$$

where  $n$  and  $n+1$  mark the current and the next time step, respectively.

### 3.2.2 Convective term

The spatial integral of the convective term is approximated using the Gauss divergence theorem given by equation (38)

$$\int_{\tau}^{\tau+\delta\tau} \int_{V_c} \nabla \cdot (\mathbf{u}\phi) \, dV \, dt = \int_{\tau}^{\tau+\delta\tau} \sum_f (\mathbf{u}\phi)_f \cdot \mathbf{S}_f \, dt + \mathcal{O}(\|\mathbf{d}\|^2). \quad (40)$$

As an example discretization, an implicit first-order accurate Euler discretization scheme is used here for the approximation of the temporal integral, starting with the Crank-Nicolson discretization and then assuming constant face-centered values over the time step  $\delta\tau$

$$\int_{\tau}^{\tau+\delta\tau} \sum_f (\mathbf{u}\phi)_f \cdot \mathbf{S}_f \, dt = \delta\tau \sum_f \phi_f^{n+1} F_f^n + \mathcal{O}(\delta\tau^2) + \mathcal{O}(\|\mathbf{d}\|^2), \quad (41)$$

where  $F_f$  is the volumetric flux over a face  $f$ , i.e.

$$F_f = \mathbf{u}_f \cdot \mathbf{S}_f. \quad (42)$$

Disregarding differences in the mass flux  $F_f^n$  and  $F_f^{n+1}$  leads to the first-order accurate in time and second-order accurate in space, unconditionally numerically stable [61] implicit Euler temporal discretization.

### 3.2.3 Diffusion term

The discretization of the diffusion term using the first-order implicit Euler scheme is the same as for the convective term, and results in

$$\int_t^{t+dt} \int_{V_c} \nabla \cdot (\gamma \nabla \phi) dV dt = \delta\tau \sum_f \gamma (\nabla \phi)_f^{n+1} \cdot \mathbf{S}_f + \mathcal{O}(\delta\tau^2) + \mathcal{O}(\|\mathbf{d}\|^2). \quad (43)$$

### 3.2.4 Source terms

If  $S = S_\phi(\phi)$ , the unstructured **FVM** requires the source term to be linearised [101, 61, 68]. The reason lies in the computational complexity of solution algorithms for *nonlinear* algebraic equation systems. Solution algorithms for nonlinear equation systems are replaced by iterative solution algorithms for *linear* algebraic systems, so all non-linearities are dealt with by iteration.

The linearised source term is defined as

$$S_\phi(\phi) = S_e + S_c \phi \quad (44)$$

and discretized as

$$\int_\tau^{\tau+\delta\tau} \int_{V_c} S_\phi(\phi) dV dt \approx \delta\tau (S_e |V_c| + 0.5 S_c |V_c| \phi_c^n) + \delta\tau 0.5 S_c |V_c| \phi_c^{n+1}. \quad (45)$$

The terms are grouped this way to separate the explicit from implicit contributions. Unstructured **FVM** is used to approximate solutions of technical problems, where computational efficiency plays a major role because of the large problem sizes. Fast and iterative solvers for sparse linear equation systems have been developed for this purpose [116]. As a consequence, the non-linear dependence in  $S_\phi(\phi)$  is to be linearised.

### 3.2.5 Discretized generalized transport equation

The final form of the discretized generalized transport equation is given by equations (40), (41), (43) and (45)

$$\begin{aligned} |V_c| \left[ (\phi)_c^{n+1} - (\phi)_c^n \right] + \delta\tau \sum_f \phi_f^{n+1} F_f^n - \delta\tau \sum_f (\gamma \nabla \phi)_f^{n+1} \cdot \mathbf{S}_f = \\ = \delta\tau (S_e |V_c| + 0.5 S_c |V_c| \phi_c^n) + \delta\tau 0.5 S_c |V_c| \phi_c^{n+1}, \end{aligned} \quad (46)$$

for each cell  $\mathbf{C}_c$  of the computational domain  $\Omega_h$ . Assembling equation (46) for each  $\mathbf{C} \in \Omega_h$  would involve performing the calculation of face-centered values twice, since

$$\forall \mathbf{F}_f \in \mathbf{C}_c \exists! \mathbf{C}_d \in \Omega_h : \mathbf{C}_c \cap \mathbf{C}_d = \mathbf{F}_f. \quad (47)$$

To halve the number of calculations, the *owner-neighbor* face-cell addressing is introduced to the unstructured mesh:

$$\forall f \in \mathbf{M}_f \exists (N_f, P_f) : P_f < N_f, P_f, N_f \in \{1..|\Omega_h|\}, \quad (48)$$

where  $\mathbf{M}_f$  is the list of all faces in the mesh. The discretization calculations are then performed over faces and not over cells. This kind of addressing is shown in figure 3.

### 3.2.6 Interpolation

Equation (46) contains unknown face-centered values at the new iteration (time) step  $(\phi_f^{n+1})$ . The face centered value is interpolated from the cell values

$$\phi_f = w_{P_f} \phi_{P_f} + w_{N_f} \phi_{N_f}. \quad (49)$$

The linear interpolation by the Central Differencing Scheme (**CDS**) results in

$$\phi_f = w_l \phi_{P_f} + (1 - w_l) \phi_{N_f}, \quad (50)$$

where the linear interpolation weight is defined as

$$w_l = \frac{\|\mathbf{x}_f - \mathbf{x}_{N_f}\|}{\|\mathbf{x}_{P_f} - \mathbf{x}_{N_f}\|}. \quad (51)$$

The interpolation introduces an additional error into the equation discretization process. An overview of different interpolation schemes for unstructured meshes together with their error analysis is available in [61, 68]. Stability and convergence analysis of such schemes can also be found in [55].

### 3.2.7 Boundary conditions

What is left in order to finish the discretization of equation (46) are the conditions at the boundary of  $\Omega_h$ . For a boundary face  $b$ , second-order accurate Neumann zero-gradient and fixed gradient boundary conditions are defined using a linear approximation of  $\phi(\mathbf{x})$  within the cell by Taylor series according to

$$\phi_c = \phi_b + \nabla\phi_b \cdot (\mathbf{x}_c - \mathbf{x}_b) + \mathcal{O}((\mathbf{x}_c - \mathbf{x}_b) \cdot (\mathbf{x}_c - \mathbf{x}_b)). \quad (52)$$

From equation (52), when  $\nabla\phi_b = \mathbf{0}$ , the boundary value is defined as

$$\phi_b = \phi_c - \mathcal{O}((\mathbf{x}_c - \mathbf{x}_b) \cdot (\mathbf{x}_c - \mathbf{x}_b)) \quad (53)$$

and for a fixed gradient  $\nabla\phi_b = \mathbf{g}$ , it is defined as

$$\phi_b = \phi_c - \mathbf{g} \cdot (\mathbf{x}_c - \mathbf{x}_b) - \mathcal{O}((\mathbf{x}_c - \mathbf{x}_b) \cdot (\mathbf{x}_c - \mathbf{x}_b)). \quad (54)$$

## 3.3 SOLUTION OF THE LINEAR ALGEBRAIC EQUATION SYSTEM

Applying equation (50) and setting the boundary face values according to section 3.2.7, equation (46) can be written as

$$a_c\phi_c + \sum_{N \in N_{c,c}} a_N\phi_N = S_c, \quad (55)$$

where  $N$  is the index of the surrounding cells and  $S_c$  is the source term that contains all the terms pertinent to the current time step  $n$ . Assembling equation (55) for every cell  $c$  results in a system of linear algebraic equations of type

$$\mathbf{Ax} = \mathbf{b}. \quad (56)$$

The interpolation stencil defined by equation (50) shows that only the immediate adjacent cells take part in the equation discretization for every cell  $c$ . Since equation (55) is assembled for every  $c$ ,  $A$  will be a sparse matrix of dimensions  $|\Omega_h| \times |\Omega_h|$ . The diagonal of the matrix stores  $a_c$  coefficients for each cell  $c$ . The form of equation (55), and therefore the quadratic form of the matrix  $A$ , is determined by the indices  $N_f$  and  $P_f$ . The cell indices are generated by the *unstructured* mesh generation algorithm. If the  $|N_f - P_f|$  is small for each  $f$ , the matrix coefficients will be packed closer to the matrix diagonal. Because  $|\Omega_h|$  is large, the solution of such a linear algebraic equation system is obtained approximatively, using iterative solution algorithms or sparse systems. The reader is directed to [116] and [124] for more information on this topic.

### 3.3.1 *Source term linearization*

A condition on equation (45), namely

$$S_c < 0, \tag{57}$$

guarantees that the source term linearisation will not spoil the weak diagonal dominance of  $A$ . When  $S_c > 0$ , explicit linearisation of the source term should be used. Linearisation of source term functions that are at least  $C^1$  can be done using Taylor series approximation.

The iterative approach to dealing with nonlinear equations assembles the system coefficients using previous field values, computes the approximative solution of the algebraic system, and updates the coefficients. This can be repeated a prescribed number of times, or up to a specified tolerance.

## **Part II**

# **Geometrical Volume-of-Fluid Method**





---

## METHOD OVERVIEW

---

This chapter covers a brief introduction of the geometrical **VoF** method that facilitates easier understanding of both the survey of the relevant literature covered in chapter 5 and the proposed method covered in chapters 5 and 6. Other sources with information similar to what is presented in this chapter are available. A very motivating and interesting overview of the geometrical **VoF** method was written by Kothe [70]. Tryggvason, Scardovelli, and Zaleski [139] have published a book on **DNS** of gas-liquid multiphase flow with a large part devoted to the geometrical **VoF** method. Zhang [151, 152] has described the geometrical **VoF** method in rigorous detail and provided theoretical background that connects different geometrical **VoF** method categories.

The discretization of equation (9) by the proposed geometrical **VoF** method is based on a so-called *dimensionally un-split* approach to geometrical volume fraction advection. The dimensionally un-split algorithm assumes no alignment of cell face normal vectors with the coordinate axes. An alternative, *dimensionally split* geometrical approach to solving equation (9) is characteristic for structured meshes. The dimensionally un-split advection has been shown to improve the overall accuracy of the solution in terms of all volume fraction advection errors. For these reasons, and because the proposed geometrical **VoF** method approximates the solution of the volume fraction on unstructured meshes, where there is no alignment of face area normal vectors, dimensionally split algorithms are not considered.

Two method categories of the un-split geometrical **VoF** method have been developed so far: the Lagrangian tracing / Eulerian remapping (**LE**) flux-based method and the **LE** cell-based (re-mapping) method. Both methods consist of two sub-algorithms: the *interface reconstruction* and the *volume fraction advection* algorithm. The geometrical interface reconstruction algorithm numerically approximates the map  $\mathcal{M}_{\alpha,I}$  required for equation (9). The volume fraction advection algorithm numerically approximates the r.h.s of equation (9).

#### 4.1 LAGRANGIAN / EULERIAN FLUX-BASED METHOD

##### 4.1.1 Discretization of the volume fraction equation

At this point we assume that the solution domain has been discretized as outlined in section 3.1 into non-verlapping control volumes (cells)  $V_c$ . The exact volume fraction equation (9) is restated here for clarity:

$$\alpha_{i,V}(\tau + \delta\tau) = \alpha_{i,V}(\tau) - \frac{1}{|V|} \int_{\tau}^{\tau+\delta\tau} \int_{\partial V} I_i(\mathbf{x}, t) \mathbf{u}(\mathbf{x}, t) \cdot \mathbf{n} d\mathbf{o} dt.$$

To study the numerical approximation of the fluid interface evolution, it is sufficient to consider two phases. In this case, a single indicator function  $I_1(\mathbf{x}, t) := I(\mathbf{x}, t)$  and volume fraction  $\alpha_{1,V} = \alpha_V$  can be used, because the indicator function and the volume fraction of the other phase are respectively given by  $1 - I(\mathbf{x}, t)$  and  $1 - \alpha_v$ . Each control volume  $V_c$  (mesh cell  $\mathbf{C}_c$ ) of the discretized domain  $\Omega_h$  is therefore associated with the volume fraction  $\alpha_c$ , so for each cell  $c$ , the exact equation (9) is given as

$$\alpha_c(\tau + \delta\tau) = \alpha_c(t) - \frac{1}{|V_c|} \int_{\tau}^{\tau+\delta\tau} \int_{\partial V_c} I(\mathbf{x}, t) \mathbf{u}(\mathbf{x}, t) \cdot \mathbf{n} d\mathbf{o} dt. \quad (58)$$

Following the time index notation given by equation (28), and recognizing the fact that  $V_c$  is the volume enclosed by a convex polyhedral cell  $\mathbf{C}_c$ , equation (58) can be rewritten as

$$\alpha_c^{n+1} = \alpha_c^n - \frac{1}{|V_c|} \sum_f \int_t^{\tau+\delta\tau} \int_{S_f} I(\mathbf{x}, t) \mathbf{u}(\mathbf{x}, t) \cdot \mathbf{n} d\mathbf{o} dt, \quad (59)$$

where  $S_f$  is the face of the polyhedral mesh cell  $\mathbf{C}_c$  shown in figure 3. Note that equation (59) is still exact, it is simply equation (9), applied on the cell control volume  $V_c$ .

A standard solution approximation for equation (59) using unstructured **FV** method covered in chapter 3 requires the integrals over finite volume faces  $S_f$  to be replaced with averaged values associated with the face centers. Equation (46) is a valid example of the unstructured **FV** method discretization for a generalized scalar transport equation. Many interpolation schemes have been developed to estimate face-centered values within the unstructured **FV** method. Some of them have been specially developed to deal with fields that contain abrupt changes in their values, like the  $\alpha_c$  field in equation (59). Even though  $\alpha_c$  is not discontinuous, its values still change from 0 to 1 within only a single layer of mesh cells. With increased mesh resolution, this *numerical jump* between 0 and 1 is more accurately resolved. All interpolation schemes used to discretize differential operators result in instabilities in the near vicinity of the jump, since their error is proportional to the spatial derivative of  $\alpha_c$ . Since  $\alpha_c$  changes abruptly within a single cell layer, all its spatial derivatives increase in magnitude with increased mesh resolution. That is why the use of interpolation schemes to evaluate  $\alpha_f$  leads to artificial smoothing and/or

dispersion of the solution [142], [55, sec. 8.8], [30], [91, ch. 12], etc. Still, interpolation based schemes have many advantages. It is known that the implementation of algebraic interpolation schemes is more straightforward than the implementation of the geometrical VoF method in 3D. Moreover, their parallelization is of the same complexity as the parallelization of the numerical method that they rely on, i.e. the FV method. Furthermore, algebraic operations are more robust than the geometric ones which must involve special case handling. For these reasons, algebraic VoF methods are still widely used for simulating two-phase flows.

The motivation behind the development of geometrical VoF methods lies in the increased overall accuracy of the interface advection problem, increased order of convergence and numerical consistency, as well as the applicability of higher values of the Courant-Friedrichs-Lewy (CFL) criterion in practice, that make the geometrical VoF method a good candidate for DNS of two-phase flows. When devised and especially implemented correctly, known problems that involve the robustness, numerical stability as well as computational efficiency can at least be solved to an acceptable level, if not solved completely.

The geometrical VoF method relies on an alternative approach: it is based on a *geometrical approximation* of the integral on the r.h.s of equation (59). The focus of the geometrical VoF method lies in the geometrical interpretation of this integral, that can be interpreted as a *fluxed phase volume*: the *volume of fluid 1* transported across the face  $S_f$  within a single time step. The spatial and temporal order of accuracy of an approximated solution to equation (59) is completely determined by the order of accuracy of the integral term approximation. Computing an integral over a single face  $S_f$  in the equation (59) in time with a first-order explicit Euler method results in

$$V_f^\alpha = - \int_{\tau}^{\tau+\delta\tau} \int_{S_f} I(\mathbf{x}, t) \mathbf{u}(\mathbf{x}, t) \cdot \mathbf{n} ds = \int_{V_f} I(\mathbf{x}, t) dx \subseteq V_f, \quad (60)$$

where  $V_f$  is the *flux volume* computed by tracing the face  $f$  backwards along discrete Lagrangian trajectories. Point displacements along discrete trajectories integrated with the first-order accurate rectangle quadrature  $-\mathbf{u}_p \delta t$  are shown in figure 4. The integral over  $V_f$  has a physical interpretation: a volume of phase 1 that crosses the face  $S_f$  over the time interval  $\delta t$ . This volume is defined in this text as the *phase flux volume*  $V_f^\alpha$  and it is shaded in figure 4.

The phase flux volume is a subset of the *flux volume* filled with phase 1. If  $\delta t \rightarrow 0$ , equation (60) represents the exact phase flux volume. In turn, a first-order accurate in time flux volume is defined as

$$V_f = - \int_{\tau}^{\tau+\delta\tau} \int_{S_f} \mathbf{u}(\mathbf{x}, t) \cdot \mathbf{n} ds dt = - \int_{S_f} \delta t \mathbf{u}(\mathbf{x}, t) \cdot \mathbf{n} ds + \mathcal{O}(\delta t). \quad (61)$$

A temporal discretization error is introduced at this point by approximating both the *length* and *direction* of the displacements along the backward discrete Lagrangian trajectories, determined by the quadrature rule. The velocity field  $\mathbf{u}(x, t)$  that generally varies in space will create *ruled surfaces* with each face edge

$\mathbf{e}_{f,i}$  that is swept along the discrete Lagrangian trajectories. The ruled surfaces created this way are therefore in a general case nonlinear.

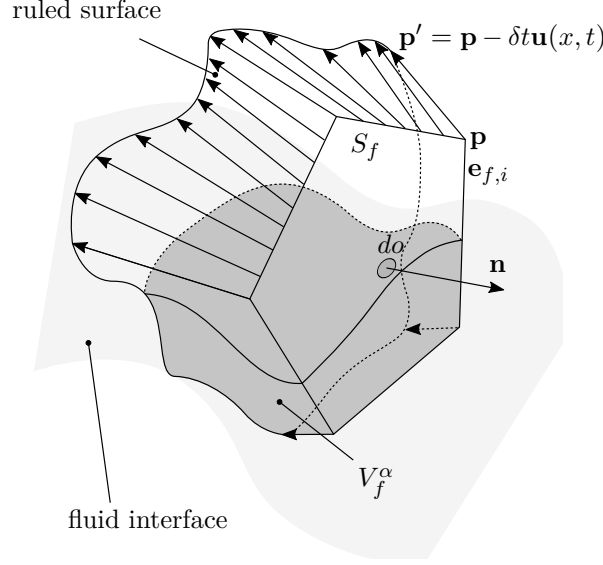


Figure 4: Schematic representation of a phase flux volume  $V_f^\alpha$  constructed as an intersection between the fluid interface and the flux volume  $V_f$  with nonlinear ruled surfaces.

Apart from the time step approximation, surfaces of the flux volume as well as the fluid interface must be approximated. Ruled surface approximation is necessary in order to approximate the integral terms in equation (60). Interface approximation is necessary in order to approximate  $I(x, t)$ , a requirement for equation (60).

*Nonlinear surface approximation* and related Boolean operations can hypothetically be applied to computing the phase flux volume. The field of Computational Solid Geometry (**CSG**) deals with such operations, that represent the core functionality of Computer Aided Design (**CAD**) software. However, high computational costs are involved in nonlinear surface approximation and intersection. The costs are negligible within the **CAD** software that allows real-time manipulation of dozens of such nonlinear geometrical solid objects. Note however that such intersections are executed once per each finite volume face that has at least one neighboring cell that contains the fluid interface by the geometrical **VoF** method. In a general case, meshes used for two-phase simulations contain millions of cells per Central Processing Unit (**CPU**) core on modern computer architectures. Even though only a percentage of the total number of cells contain the fluid interface, the number of required phase flux volume calculations is still too large for costly 3D non-linear **CSG** intersection operations.

Therefore, the dimensionally un-split geometrical **VoF** method relies on *surface linearisation* for approximating ruled surfaces as well as the fluid interface. Different approaches for the ruled surface linearisation have been proposed so far and are covered in detail in section 5.2. At this point it is sufficient to assume that the ruled surfaces are linearised by sets of triangles: they are *triangulated*.

As is the case for the ruled surface linearisation, different approaches to the linearisation of the fluid interface have been developed so far as well. Details on

contemporary interface linearisation algorithms are covered in section 5.1. The prevalent linearisation approximates the interface as a single plane per control volume  $V_c$ , so this approximation is adopted here to describe how the geometrical VoF method discretizes the map  $\mathcal{M}_{\alpha,I}$ . Therefore, a nonlinear interface is approximated as plane  $\Pi_c$  in each *interface cell*, defined by the interface orientation vector  $\mathbf{n}_c$  and the position vector  $\mathbf{p}_c$

$$\Pi_c = \Pi(\mathbf{n}_c, \mathbf{p}_c) = \{\mathbf{x} : \mathbf{n}_c \cdot (\mathbf{x} - \mathbf{p}_c) = 0\}. \quad (62)$$

From the definition of  $\alpha_{i,v}$  by equation (6) it is clear that the fluid interface resides in interface cells, where  $0 < \alpha_c < 1$ . Figure 5 shows the aforementioned linearisation of the nonlinear ruled surfaces as well as the interface from figure 4, along with the corresponding phase flux volume  $V_f^\alpha$ .

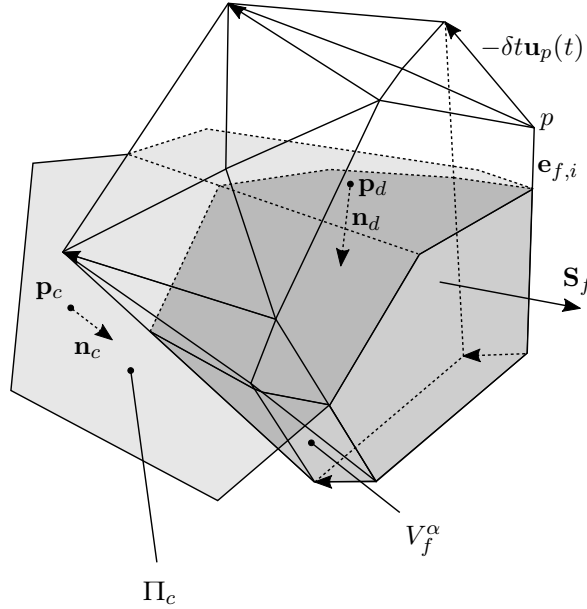


Figure 5: A temporally first-order accurate approximation of the phase flux volume.

The interface plane  $\Pi_c$  defines a *positive interface half-space*

$$\mathcal{H}_c = \{\mathbf{x} : \mathbf{n}_c \cdot (\mathbf{x} - \mathbf{p}_c) > 0\}. \quad (63)$$

This in turn results in an approximation of the phase indicator function defined by equation (1) as

$$I_c(\mathbf{x}, t) = \begin{cases} 1 & \mathbf{x} \in \mathcal{H}_c, \\ 0 & \text{otherwise.} \end{cases} \quad (64)$$

The volume fraction  $\alpha_c$  used in the volume fraction equation (59) is computed as

$$\alpha_c = \frac{1}{|V_c|} |\mathcal{H} \cap V_c|, \quad (65)$$

representing a linear approximation of the map  $\mathcal{M}_{I,\alpha}$ . Considering the face  $S_f$ , we can define  $I(\mathbf{x}, t)^{+,-} = \lim_{h \rightarrow 0^+} I(\mathbf{x} \pm h \mathbf{S}_f)$ , where  $\mathbf{S}_f$  is the surface area normal vector of the face  $S_f$  and  $\mathbf{x} \in S_f$ . Because the indicator function  $I(\mathbf{x}, t)$  is continuous across  $S_f$  along the direction of its normal vector, the following can be stated

$$\lim_{\delta\tau \rightarrow 0^+} \int_{\tau}^{\tau+\delta\tau} \int_{S_f} I(\mathbf{x}, t)^+ \mathbf{u} \cdot \mathbf{n} ds - \int_{\tau}^{\tau+\delta\tau} \int_{S_f} I(\mathbf{x}, t)^- \mathbf{u} \cdot \mathbf{n} ds = 0. \quad (66)$$

However, the linear approximation of the indicator function  $I_c$  within  $V_c$  using equation (64), makes the interface *piecewise* planar. An approximation that is piecewise-planar introduces a discontinuity in the  $I_c$  across the face  $S_f$ . Considering equation (66), the discontinuity introduced by the piecewise planar approximation therefore leads to an error in computing the fluxed phase volume.

The flux volume  $V_f$  may intersect multiple cells that contain the interface, as shown in figure 5 for cells  $c$  and  $d$ , where the flux volume is intersected with two positive half-spaces defined by  $(\mathbf{n}_c, \mathbf{p}_c)$  and  $(\mathbf{n}_d, \mathbf{p}_d)$ , respectively. As a consequence, the phase flux volume is separated into a sum of *phase flux volume contributions* from the corresponding cells that intersect the flux volume

$$V_f^\alpha = \int_{V_f} \alpha dx \approx \sum_{c \in \mathcal{C}_f} \int_{V_{f,c}} \tilde{\alpha}_c dx = \sum_{c \in \mathcal{C}_f} V_{f,c}^\alpha, \quad (67)$$

where  $\mathcal{C}_f$  is the set of cells that intersect the flux volume  $V_f$  into individual flux volume contributions  $V_{f,c}$

$$V_{f,c} = V_f \cap V_c. \quad (68)$$

and

$$V_{f,c}^\alpha = \mathcal{H}_c \cap V_{f,c}. \quad (69)$$

In other words, the phase flux contribution from a cell  $c$  is a result of the intersection of the flux volume  $V_f$  and  $V_c$  resulting in the flux volume contribution  $V_{f,c}$ , that is subsequently intersected with the positive half-space  $\mathcal{H}_c$ . Showing this schematically is difficult in three dimensions, therefore the schematic representation of the phase flux contribution calculation is shown in 2D in figure 6.

The flux volume cell neighborhood  $\mathcal{C}_f$  in figure 6 consists only of 2 cells that have a volume intersection with  $V_f$ :

$$\mathcal{C}_f = \{\mathbf{C} : \text{volume}(\mathbf{C} \cap V_f) > 0\}. \quad (70)$$

This reduction in size of  $\mathcal{C}_f$  significantly reduces the computational complexity. Each flux volume contribution  $V_{f,c}$  is computed as an intersection using equation (68) and is shaded differently in the left image of figure 6. From each flux volume contribution  $V_{f,c}$  a *phase* flux volume contribution is computed using equation (69) and is shaded differently in the right image of figure 6. After all phase flux volume contributions are computed, the total phase flux volume that crosses the face  $f$

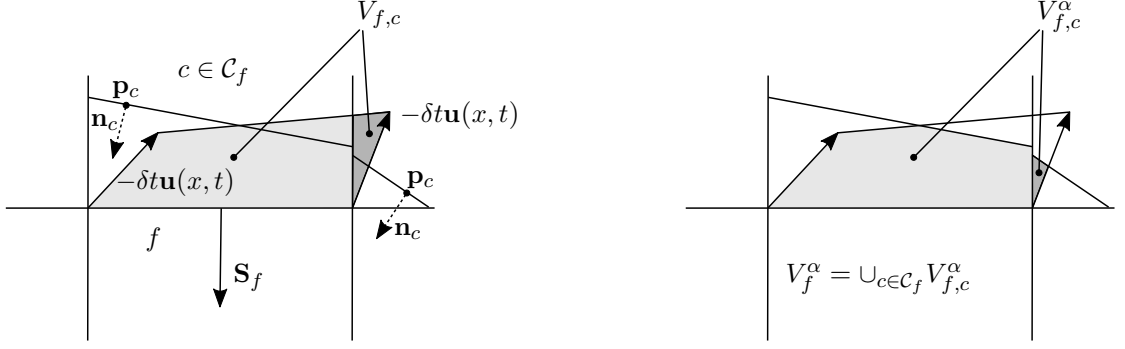


Figure 6: A 2D schematic representation of the phase flux volume  $V_f^\alpha$  calculation. The image on the left schematically represents the information required for geometrically approximating the flux volume  $V_f$ . The image on the right side shows the phase flux volume contributions  $V_{f,c}^\alpha$  computed from the flow configuration shown on the left.

is computed using the sum in equation (67). This is a final step required for the piecewise-linear geometrical approximation of equation (59), that takes on the following fully *geometrically* discretized form:

$$\alpha_c^{n+1} = \alpha_c^n - \frac{1}{|V_c|} \sum_f V_f^\alpha + \mathcal{O}(\|\mathbf{d}\|^2) + \mathcal{O}(\delta t^p) \quad (71)$$

that is second-order accurate in space provided a second-order accurate piecewise linear approximation of the interface is used, and has  $p$ -th order of accuracy in time. The order of accuracy of the temporal integration  $p$  is determined by the order of accuracy of the quadrature rule used to integrate discrete Lagrangian displacements. Geometrical calculations involved in the covered discretization steps of the volume fraction equation (59) separate two distinct computational sub-algorithms of the geometrical **VoF** method: the *interface reconstruction* and the *volume fraction advection* algorithm, introduced in the following sections.

#### 4.1.2 Interface reconstruction

In section 4.1.1 it is assumed that a linear approximation of the fluid interface is available in each interface cell. The geometrical **VoF** method *reconstructs* the linear approximation of the fluid interface from the volume fraction field  $\alpha_c$  by reconstructing the interface plane (half-space)  $\mathcal{H}_c$  in each interface cell such that

$$\alpha_c = \frac{1}{|V_c|} \int_{V_c} I(\mathbf{x}, t) dx = \frac{|\mathcal{H}_c(\mathbf{x}, t) \cap V_c|}{|V_c|}. \quad (72)$$

In order to reconstruct the half-space, the reconstruction algorithm obviously relies on an available discrete volume fraction field  $\alpha_c$ . In the initial time step, these values are a result of an intersection between an *initial interface* and the mesh  $\Omega_h$ . The initial interface can be modeled using an indicator function  $\tilde{I} = I(\mathbf{x}, t)$ .



Alternatively, the initial interface can also be defined as a *computational mesh* that represents a discretely approximated interface, a so-called *interface mesh*.

$$\tilde{I} = \Gamma_h. \quad (73)$$

In this case, even though it is geometrically approximated by a surface or a volume mesh, the initial interface is considered exact with respect to the reconstructed piecewise linear interface that contains reconstruction errors. In both cases, the discrete *initial*  $\alpha_c$  is given as

$$\alpha_c(t = t_0) = |\tilde{I} \cap \Omega_h|. \quad (74)$$

When the exact interface is defined using an explicit continuous function, schematically shown on figure 7(a), the calculation of the actual discrete volume fraction still requires an approximation of the integral

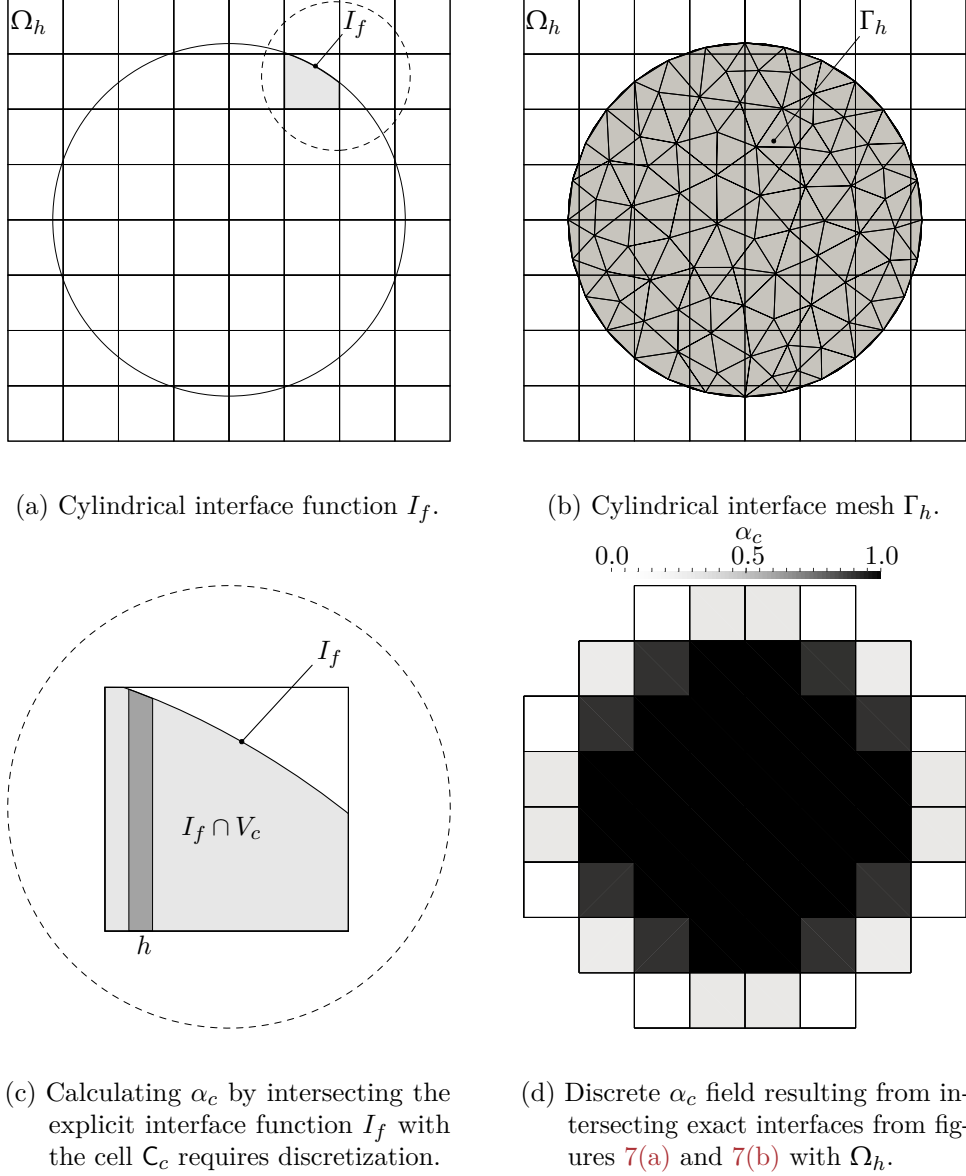
$$\alpha_c = \frac{1}{|V_c|} \int_{V_c} I(\mathbf{x}, t) dx, \quad (75)$$

for each interface cell because an exact evaluation of equation (75) is only possible for simple convex polyhedral volumes  $V_c$  and simple functions  $I(\mathbf{x}, t)$ . The interface cell is discretized into small intervals  $h$  as shown in figure 7(c), so that the integral can be approximated for an interface function  $I_f$ . Because of the misalignment of the face area normal vectors with coordinate axes, such integration becomes difficult for polyhedral cells. That is why in this work a geometrical exact interface (interface mesh) is used to set the initial  $\alpha_c$  values by intersecting it with the solution domain  $\Omega_h$ . Note that in this case, the resolution of the geometrical exact interface determines the absolute accuracy of the reconstruction algorithm: a reconstructed interface can only be as accurate as its geometrically modeled exact counterpart.

Consider as an example the interface mesh on  $\Gamma_h$  shown in figure 7(b) and the corresponding  $\alpha_c$  field shown in figure 7(d). In order to compute  $\alpha_c$ , geometrical intersections between the cells of  $\Gamma_h$  and the cells of  $\Omega_h$  are performed. The geometrical intersections avoid the complexity required by equation (75) from computing  $h$  for cell faces of arbitrary shapes. Note that because an integral of an exact indicator function is approximated, such initialization of the  $\alpha_c$  contains numerical integration errors. This is so regardless of the chosen integration method.

The intersection based approach requires an intersection between each two polyhedrons from  $\Gamma_h$  and  $\Omega_h$  respectively. Such an implementation is of complexity  $\mathcal{O}(|\Omega_h||\Gamma_h|)$ , resulting in very long computational times for large cases. The quadratic complexity can readily be optimized using intersection reduction algorithms based on spatial partitioning such as algorithm 21 or based on intersection reduction tests. One advantage of the intersection based approach compared to an explicit indicator function  $I(\mathbf{x}, t)$  is the ability to intersect two solution domains of arbitrary geometrical complexity with each other. The accuracy of the calculated  $\alpha_c$  is determined by the accuracy of the geometrical intersections. In this work,



Figure 7:  $\alpha_c$  field initialization.

individual geometrical intersections are accurate up to  $4\epsilon_t$ , where  $\epsilon_t$  is the machine tolerance.

Once the discrete volume fraction field has been initialized, the reconstruction algorithm can use it to reconstruct the piecewise linear interface.

To ensure the robustness of the method,  $\mathcal{H}_c$  is reconstructed in each interface cell up to a specified *reconstruction tolerance*

$$\epsilon_R < \alpha_c < 1 - \epsilon_R. \quad (76)$$

Values for  $\epsilon_R$  are chosen within  $[1e-09, 1e-06]$  in the literature, in this work  $1e-09$  is used. To define the half-space  $\mathcal{H}_c$ , the reconstruction algorithm needs to determine its position  $\mathbf{p}_c$  and orientation  $\mathbf{n}_c$ . Therefore, the reconstruction algorithm is further subdivided into *interface orientation* and *interface positioning* algorithms.

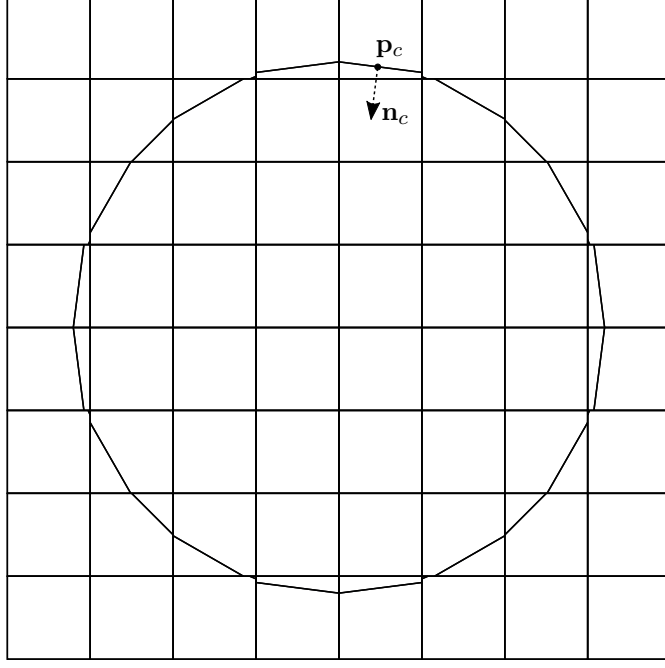


Figure 8: Reconstructed interface from the field  $\alpha_c$  shown in figure 7(d).

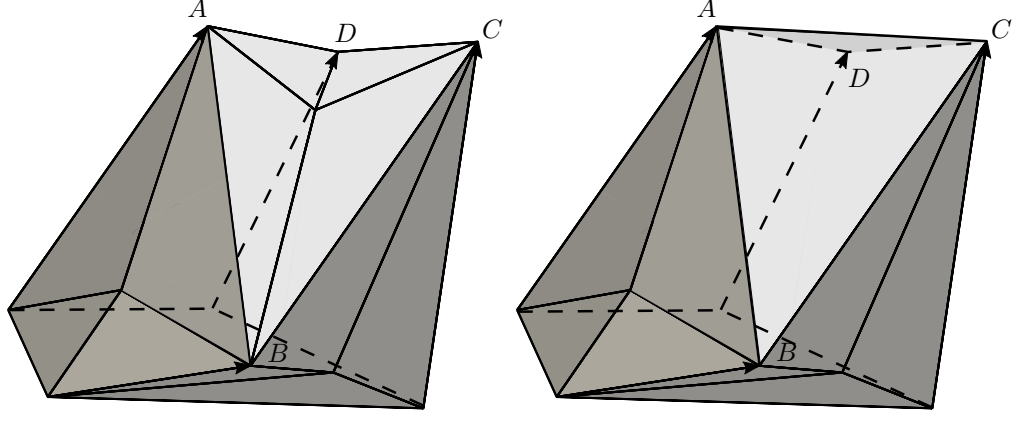
Interface orientation algorithms that have been developed so far, are aiming at accurately estimating the normal vectors  $\mathbf{n}_c$  based on the field values  $\alpha_c$ . Once the interface orientation is complete, it is followed by interface positioning, that computes a unique plane position  $\mathbf{p}_c$  from the volume fraction value  $\alpha_c$ . The configuration of the plane  $\Pi_c$  is uniquely determined by  $\mathbf{n}_c$  and  $\mathbf{p}_c$  for all cell shapes. An overview of state-of-the-art interface orientation and positioning algorithms is presented in section 5.1. The interface orientation and positioning algorithms proposed in this work are covered in section 6.2. Figure 8 shows a reconstructed interface that approximates the cylindrical interface mesh  $\Gamma_h$  shown in figure 7(b). Such a reconstructed interface is then used by the advection algorithm to update the volume fraction field in the new time step.

Once the piecewise linear interface has been reconstructed, it is utilised by the volume fraction advection algorithm to solve equation (71).

#### 4.1.3 Volume fraction advection

The description of the volume fraction discretization covered in section 4.1.1 provides a concise overview of the geometrical VoF method. Section 4.1.1 however only introduces aspects of the geometrical VoF method that are necessary to obtain an overview of the solution approach. The geometrical approximation of the flux volume  $V_f$  and the volume calculation of geometrical objects that represent the flux volumes require careful attention. When not considered carefully, they may lead to errors in volume conservation, stability and absolute accuracy of the solution.

A linearisation of the ruled surfaces in the form of a triangulation, shown in figure 5 is not the only linearisation procedure. Triangulations of convex sets allow



(a) Barycentric surface triangulation.

(b) Oriented surface triangulation.

Figure 9: A complex flux polyhedron with non-planar and non convex ruled surfaces. Using different triangulations for a chosen face (cap polygon  $ABCD$ ) of this flux polyhedron result in different flux volume  $V_f$  magnitude.

graph-theory analysis that can yield the number of possible triangulations, as well as their mutual dependence [32]. Existing and newly proposed flux volume triangulations are covered in section 6.3. For now, consider figure 9: it shows two different ruled surface triangulations, leading to different flux volumes  $V_f$ .

The *non-planar* surface defined by point sequence  $(A, B, C, D)$  can be triangulated using the surface centroid (figure 9(a)) given by equation (20). Alternatively, it can be triangulated with either of the two diagonals  $AC$ , or  $BD$ . It is very important to note that *different ruled surface approximations yield different flux volumes*. The mass conservation equation for an incompressible fluid is formulated in terms of the divergence free condition of the velocity

$$\nabla \cdot \mathbf{u}(\mathbf{x}, t) = 0, \quad (77)$$

which, discretized using the unstructured **FV** method, becomes

$$\int_{\tau}^{\tau+\delta\tau} \int_{V_c} \nabla \cdot \mathbf{u} dx = \sum_f \int_{\tau}^{\tau+\delta\tau} \mathbf{u}_f(t) \cdot \mathbf{S}_f dt = 0, \quad (78)$$

where  $\mathbf{u}_f$  is the face-centered velocity and  $\mathbf{u}_f(t) \cdot \mathbf{S}_f = F_f(t)$  is the face-centered volumetric flux. Equation (78) imposes a constraint on the flux volume given by equation (61):

$$\int_{\tau}^{\tau+\delta\tau} \int_{S_f} \mathbf{u}(\mathbf{x}, t) \cdot \mathbf{n} do dt = V_f = \int_{\tau}^{\tau+\delta\tau} \mathbf{u}_f(t) \cdot \mathbf{S}_f dt. \quad (79)$$

The condition given by equation (79) is not upheld, because the flux volume is generated by displacing the points  $\mathbf{p}_i$  of the face  $f$  along discrete Lagrangian trajectories, followed by the already discussed linear approximation of ruled surfaces. These approximations require the flux volume  $V_f$  to be *corrected for volume*

*conservation*, whose magnitude will depend on the shape of the geometrical flux volume.

To emphasize the difference between the geometric and the algebraic flux volume, the shape of the *geometrical flux volume* depends on:

- different forms of the discrete Lagrangian trajectories resulting from different temporal integration schemes,
- the choice of the approximation for ruled surfaces generated by the discrete Lagrangian trajectories,
- optional geometrical flux simplification algorithms.

For example, figure 9 depicts a geometrical flux volume constructed with a first-order accurate Euler back-tracing of the base points, an *edge triangulation* used to triangulate the non-planar ruled surfaces and a *flux triangulation* used for volume computation. An alternative linearisation of the ruled surfaces could result in a *flux polyhedron* to be constructed. To avoid confusion between different forms of the geometrical flux volumes, a single term *geometrical flux volume* is used in this text when the actual geometrical form is not of importance. When it is important to distinguish if the geometrical flux volume is in fact a polyhedron or a triangulation, then those specific terms are used.

Once the geometrical flux volume has been corrected for volume conservation, the next step of the advection algorithm is to compute the phase flux volume by computing individual phase flux volume contributions  $V_{f,c}^\alpha$  given by equation (69). The parallelization of the Eulerian flux-based geometrical **VoF** method using the domain decomposition and message passing approach is trivial: outflow phase flux volume magnitudes are computed on the inter-process boundary and are consequently exchanged between processes.

To summarize, the task of the volume fraction advection algorithm consists of the geometrical approximation of the flux volume, its correction for volume conservation and finally the calculation of the phase flux contributions  $V_{f,c}^\alpha$  follows, by performing geometrical intersections of the corrected geometrical flux volume with surrounding interface cells and half-spaces.

## 4.2 LAGRANGIAN BACKWARD TRACING / EULERIAN REMAPPING

The Lagrangian / Eulerian flux-based geometrical **VoF** method is derived by modeling the motion of the fluid interface as it passes through a *fixed* volume  $V$ . The Lagrangian tracing / Eulerian remapping (**LE**) method observes the motion of a *moving* control volume  $V$  that moves relative to the interface. Formal derivation of the **LE** and the derived equivalence with the Eulerian flux-based method is given by Zhang [151], who shows that the flux-based equation (59) is equivalent to

$$\alpha_c^{n+1} = \frac{1}{|V_c^{n+1}|} \int_{V_c^{n+1}} I(x, t) dx \quad (80)$$

which obtains the following discrete form, based on geometrical approximations covered in section 4.1.1

$$\alpha_c^{n+1} \approx \frac{1}{|V_c^{n+1}|} \sum_{d \in \mathcal{C}_c} |\mathcal{H}_{d,\alpha}^n \cap \mathbb{C}_c^{n+1} \cap \mathbb{C}_d^n| \quad (81)$$

Equation (80) states that the volume fraction in the volume  $V_c$  in the new time step  $t + \delta t \equiv n + 1$  (following equation (28)) is given by moving the volume  $V_c$  backwards, relative to a fluid interface fixed in space over  $\delta t$ . The volume is traced backwards along the Lagrangian trajectories given by the reversed velocity field  $-\mathbf{u}(x, t)$ . Therefore, the volume fraction at  $t + \delta t$  is the integral of the indicator function over the displaced control volume  $V_c^{n+1}$ . The discrete form of equation (80), equation (81) traces backwards every cell that contains an interface, or is a neighbor of an interface cell,  $\mathbb{C}_c \in \Omega_h$  over  $\delta t$  using the discrete Lagrangian trajectories ( $\delta_t^{tr} \mathbf{x}$ ) and then intersects each displaced cell ( $\mathbb{C}_c^{n+1}$ ) with surrounding interface half-spaces from the current time step  $\mathcal{H}_{d,\alpha}^n$  and their corresponding interface cells  $\mathbb{C}_d$ . The cell-point cell neighborhood  $\mathbb{C}_c$  does not change in time with the motion of the mesh, as the mesh motion does not change the connectivity of the mesh.

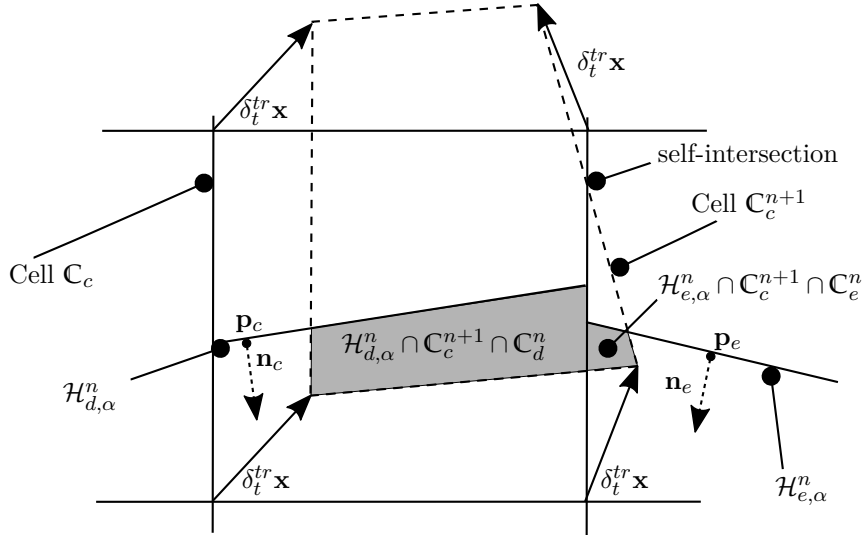


Figure 10: A schematic diagram of the discrete Lagrangian tracing / Eulerian remapping (LE) geometrical VoF method.

Figure 10 depicts schematically the Lagrangian *backward tracing* / Eulerian re-mapping (LE cell-based VoF method) version of the geometrical VoF method, because the cell  $\mathbb{C}_c^{n+1}$  is swept backwards with respect to the flow velocity  $\mathbf{u}(x, t)$ . Numerical boundedness is guaranteed by the LE geometrical VoF method, as the sum term in equation (81) used to compute the geometrical volume contributions to the cell  $\mathbb{C}_c^{n+1}$  can either leave the cell empty or full: it is not possible to obtain  $\alpha_c > 1$  or  $\alpha_c < 0$  when equation (81) is used. However, global volume conservation is not imposed by equation (81) for a solenoidal velocity field  $\nabla \cdot \mathbf{u}(x, t) = 0$ , when the point displacements result from integrating velocities that are interpolated at cell corner points (mesh points), because an interpolation error causes the velocities at mesh points to no longer be discretely divergence free. Different LE methods

discussed in the literature survey in section 5.2 were proposed with different solutions for the volume conservation problem. The parallel implementation of the **LE** method is more complicated than that of the Eulerian flux-based geometrical **VoF** method. If the cell  $\mathbf{C}_e$  in figure 10 is handled by the different process than cell  $\mathbf{C}_c$ , in order to compute  $\mathcal{H}_{e,\alpha}^n \cap \mathbf{C}_c^{n+1} \cap \mathbf{C}_e^n$ , the cell image  $\mathbf{C}_c^{n+1}$  must be sent to the other process, and the corresponding volume contribution must be received from the other process. An alternative approach of exchanging  $(\mathcal{H}_{e,\alpha}^n, \mathbf{C}_e^n)$  results in incomparably worse parallel efficiency because the number of exchanged messages is doubled since  $\mathbf{C}$  and  $\mathcal{H}$  pairs contain different geometrical models. Additionally, message sizes are much larger than in the former parallelization approach, because the process  $p$  that contains  $\mathbf{C}_c^{n+1}$  does not have the information about which cells  $\mathbf{C}_e^n$  of the process  $q$  are intersected by  $\mathbf{C}_c^{n+1}$ . Therefore, all cells  $\mathbf{C}_e^n$  adjacent do the inter-process boundary would have to be exchanged. This, in turn, results in incomparably larger messages than exchanging only those  $\mathbf{C}_c^{n+1}$  whose single point crosses the inter-process boundary and their respective volume contributions.

Regardless of the fact that the **LE** geometrical **VoF** method does not inherently ensure volume conservation and its parallel implementation using the domain decomposition approach is both more complicated and it incurs larger parallel communication costs, there are strong reasons why the **LE** geometrical **VoF** methods are being actively researched.

The complexity of the Eulerian flux-based geometrical **VoF** method increases significantly in three dimensions. To ensure convergence, volume conservation and numerical stability, Eulerian flux-based methods must deal with accurate volume calculation of three dimensional polyhedra with non-planar, non-convex, possibly also self-intersecting faces, which is rarely the case for **LE** methods. Consider the self-intersection of the bow-tie shaped flux volume shown for the right face of the cell  $\mathbf{C}_c^{n+1}$  in figure 10. The naive implementation of the **LE** method does not deal with the intersection at all: in equation (81) the intersections with  $\mathbf{C}_c^{n+1}$  are used, instead of the intersections with  $V_f$  as described in section 4.1 and the cell  $\mathbf{C}_c^{n+1}$  can retain weak convexity even when the flux volumes are strongly non-convex, with non-planar faces and are self-intersecting. Weak convexity is in this respect defined as the possibility to exactly compute the volume of non-convex polyhedron by triangulating the polyhedron using its centroid as an additional triangulation point.

Because this work proposes enhancements to the Eulerian flux-based geometrical **VoF** method, the cell-based (remapping) **LE** methods are not covered in-depth. Still, contributions to the cell-based **LE** geometrical **VoF** method are described in chapter 5 and the results of the proposed Eulerian flux-based geometrical **VoF** method are compared against the recently reported **LE** results in chapter 11.

---

LITERATURE SURVEY

---

To the best of the author’s knowledge, a single series of publications on numerical error analysis of the geometrical reconstruction has been published so far with a focus on equidistant structured meshes [106, 107, 108]. In all other publications the error convergence resulting from numerical verification tests is used as a measure of accuracy for the geometrical VoF method, so this practice is adopted in the present work as well.

Following sections provide an overview of the state of the art geometrical VoF method sub-algorithms, categorized into interface reconstruction (section 5.1) and volume fraction advection (section 5.2) categories.

### 5.1 INTERFACE RECONSTRUCTION

The first implementation of the geometrical VoF method [28] employed a *piecewise linear* geometrical interface approximation PLIC. Later developments such as [56] have simplified the geometrical interface approximation to *piecewise constant* Simple Line Interface Calculation (SLIC). A detailed overview of the earliest publications on this topic can be found in [111, page 6, table 1], together with a table summarizing most important contributions. A review of reconstruction algorithms is also available in [103, 14]. A comparison between the error convergence and relative computational costs for more recent contributions is shown in table 1.

PPLIC algorithms have prevailed over SLIC algorithms because of their many advantages. A more accurate geometrical interface approximation is provided by PPLIC algorithms than by SLIC algorithms, resulting in second-order convergent curvature calculations on structured [110, 43, 105, 99] meshes. Reconstructing the interface with first-order accuracy in terms of the reconstruction error, the PPLIC algorithms enable the efficient computations and increased accuracy over a wide span of spatial scales relying on local dynamic Adaptive Mesh Refinement (AMR) [104, 9, 3]. They require a minimal amount of information from the neighboring cells for the interface reconstruction, which reduces artificial surface tension. Artificial surface tension introduced by the interface reconstruction algorithm is the artificial rounding of sharp corners that occurs during the repeated reconstruction of the evolving interface [154, 9]. The piecewise planar interface approximation supports the numerical simulation of the transport of insoluble surfactants *on the fluid interface* [59], which is not possible with the piecewise constant interface approximation given by the SLIC algorithm. The SLIC algorithms generate a substantial amount of *jetsam* (*flotsam*) [14]. Jetsam (*flotsam*) are elements of the geometrical interface that

are *artificially separated* from the interface and transported further with the flow velocity. Noh and Woodward [95] have introduced "jetsam" (jettisoned goods) and "flotsam" (floating wreckage) for artificially separated interface elements, according to Kothe et al. [69]. **PLIC** algorithms based on error minimization can be directly applied to unstructured meshes [89, 88, 37, 73, 90], since they rely on linear traversal of the surrounding cells without accessing cells in the direction of a coordinate system axis. The advantages of **PLIC** algorithms make them the prevailing choice for reconstructing the interface within the geometrical **VoF** method.

ALGORITHM	CONVERGENCE	COST
Youngs [148]	1.0-1.8,[14]	1
Mosso-Swartz, Mosso et al. [88]	2.0	3-4
<b>LVIRA</b> , Pilliod and Puckett [103]	2.0	9,[5]
<b>ELVIRA</b> , Pilliod and Puckett [103]	1.9-2.2	900,[77]
<b>CVTNA</b> , Liovic et al. [73]	2.0	50
<b>CLCIR</b> , López et al. [77]	2.0-2.11	3
<b>CIAM</b> , Scardovelli and Zaleski [120]	1.0-2.28	1
<b>LLSF</b> , Scardovelli and Zaleski [120], Aulisa et al. [14]	2.0	1.5
<b>MoF</b> , Dyadechko and Shashkov [37],[38]	2.0	7,[5]
<b>PIR</b> , Mosso et al. [90]	2.0	10,[112]

Table 1: **PLIC** reconstruction algorithms. The reconstruction error convergence order for circles and spheres are reported. For **CIAM** and **LLSF** the error convergence order is reported for an ellipse by the authors. Additional citations are listed for algorithms whose relative costs are not reported by the original authors.

Table 1 contains the order of error convergence as well as the average relative computational costs of **PLIC** algorithms. The Youngs' gradient-based algorithm is taken as the norm for the computational cost in all the referenced publications. It is important to note that the algorithm cost depends on the implementation. However, the costs reported in [5] rely on a shared software platform, which makes them more objective. Furthermore, relative costs reported in [103, 120, 73, 77] have been reported on structured Cartesian meshes. Additional search operations are required when extending those algorithms to unstructured meshes, which leads to a substantial increase in algorithmic complexity and consequently in computational costs.

From table 1, it follows that a sub-set of reconstruction algorithms can be used to accurately approximate the geometrical interface on unstructured meshes. The main constraint enforced by the algorithms on unstructured meshes is the inability to exercise access to mesh elements in a specific direction, e.g., accessing different face centers by changing their  $y$  coordinate. The topology of the unstructured mesh is covered in section 3.1.1. Algorithms that rely on more than first level of addressing experience a substantial increase in computational complexity on



unstructured meshes also in terms of algorithm parallelization using the domain decomposition and message passing approach. For example, using equation (14), to search over all points of all faces of a cell incurs a substantial number of iterations, compared to direct point access on structured meshes. On a structured mesh each cell center is defined with an index triplet  $(i, j, k)$ . Accessing a cell face requires adding  $\pm \frac{1}{2}$  to a corresponding index of the cell triplet. Directly addressing face centers of the  $i + \frac{1}{2}$  faces by changing their  $y$  coordinate results in simply increasing the third index of the face triple:  $(i + \frac{1}{2}, j, k + l)$ . This operation has a constant complexity per face of  $\mathcal{O}_f = 1$ .

On an unstructured mesh, a search operation must be performed on a per-cell basis to find the face in a specific direction. The search algorithms used to locate a face in a specific direction rely on the connections between mesh elements. Using a search algorithm that relies on edge-face connectivity information would result in a worst-case complexity per face of  $\mathcal{O}_f = N_{f,e}$  where  $N_{f,e}$  is the number of faces that share an edge  $e$  with a face  $f$ . This number is not large: for hexahedral cells  $N_{e,f} = 12$ . However, that number becomes a large constant factor that multiplies the overall complexity of an algorithm that loops over all mesh faces  $\mathcal{O} = N_f \mathcal{O}_f = 12N_f$ , where  $N_f$  is the number of all faces in the mesh. Therefore, an algorithm that requires *directed access to a specific face* has a 12 times greater access complexity on an unstructured than on a structured mesh. Moreover, the unstructured mesh data has a non-sequential structure which prevents hardware based memory access optimizations [42, sec. 9.9]. Such computational complexity restrictions must be considered when choosing a PLIC reconstruction algorithm for unstructured meshes. Consequently, if the algorithm relative cost reported in table 1 is already high on structured meshes, it can be disregarded as a candidate for unstructured meshes.

Following subsections cover candidate PLIC reconstruction algorithms that have been considered for use with unstructured meshes in the present work. In order to reconstruct the piecewise-linear (planar) geometrical approximation of the fluid interface, to each interface cell an interface normal vector  $\mathbf{n}_c$  and position vector  $\mathbf{p}_c$  are assigned. The aim of each reconstruction algorithm is to accurately compute those two parameters. Firstly  $\mathbf{n}_c$  is approximated by the *interface orientation algorithm* and secondly the interface plane is positioned by the *interface positioning algorithm* that calculates  $\mathbf{p}_c$ .

### 5.1.1 Interface orientation

All interface orientation algorithms rely on the volume fraction field  $\alpha$  to approximate  $\mathbf{n}_c$  at some step in their calculation. Some second-order convergent algorithms rely exclusively on the  $\alpha$  field, while others employ additional geometrical information to increase absolute reconstruction accuracy and error convergence.

### 5.1.1.1 Youngs' algorithm

The algorithm was originally developed by Youngs [148]. Interface normal  $\mathbf{n}_c$  is approximated using the numerical gradient of a discrete volume fraction field

$$\mathbf{n}_c = -\frac{\nabla_c \alpha}{\|\nabla_c \alpha\| + \epsilon_m}, \quad (82)$$

where  $\epsilon_m = 1e-15$  is the machine tolerance factor used to prevent division by zero away from the interface. The absolute accuracy and error convergence of the discrete gradient approximation  $\nabla_c$  is of crucial importance for the absolute solution accuracy and error convergence when solving the volume fraction equation (9) [24]. The discrete differential gradient operator  $\nabla_c$  is approximated using the unstructured FV method covered in chapter 3. However, using the FVM for gradient operator discretization is by no means a requirement for the geometrical VoF method - other discrete differential operators can be used as well. More details on the gradient operator discretization practice on unstructured meshes using FVM can be found in [91, ch. 2].

It is widely known that an accurate approximation of  $\nabla_c$  in equation (82) is essential for the overall accuracy and error convergence of the PLIC reconstruction algorithms. This implies that the gradient discretization practice on unstructured meshes should be carefully constructed.

Mavriplis [83] concludes that a wider stencil Inverse Distance Weighted Least Squares Gradient (IDWLSG) approximation delivers accurate results on equidistant hexahedral unstructured meshes. The author has applied a wider stencil gradient for a more accurate aerodynamic drag estimation on unstructured meshes. The author shows that the accuracy and convergence of the LS gradient deteriorates strongly on unstructured tetrahedral meshes.

Aulisa et al. [14] propose a gradient calculation that uses finite differences to compute the components of the volume fraction gradient at cell corners from cell centered values obtained by averaging finite difference operations. This gradient cannot be applied on unstructured meshes as it relies on directed addressing of mesh elements.

Similar to Mavriplis [83], Ahn and Shashkov [5] propose a LS minimization to estimate  $\nabla_c \alpha$  on unstructured meshes. However, they differ from the IDWLSG proposed by Mavriplis [83] in the fact that their Linear Least Squares Gradient (LLSG) does not rely on inverse distance weighting. Instead, they apply a second-order linear approximation of the volume fraction field  $\alpha$  using the Taylor series expansion from a cell  $c$ :

$$\alpha_l(\mathbf{x}) \approx \alpha_c + \nabla_c \alpha \cdot (\mathbf{x} - \mathbf{x}_c). \quad (83)$$

A volume fraction error for the cell neighborhood  $\mathcal{C}$  is defined as

$$\epsilon_l = \sum_{n \in \mathcal{C}} \left( \alpha_n - \frac{\int_{V_n} \alpha_l(\mathbf{x}) dx}{V_n} \right)^2 \quad (84)$$

and the **LS** minimization of  $\epsilon_l$  results in a  $3 \times 3$  linear algebraic system that is solved for the 3 components of  $\nabla_c \alpha$  in each cell  $c$ . The calculations required to obtain the linear algebraic system are outlined in the appendix A. Ahn and Shashkov [5] have used the gradient estimation for the  $\alpha$  field used by Garimella et al. [45] for their Arbitrary Lagrangian-Eulerian (**ALE**) method. Both methods are focused on the use of polyhedral meshes, which makes the explicit construction of a wider gradient stencil as proposed by Mavriplis [83] redundant. This point is adequately illustrated by figure 11: the cell-face-cell connectivity defines a convex cell stencil for the polygonal cell, and a cross stencil for the hexagonal cell. In three dimensions the same conclusion applies. Regularly shaped polyhedral cells have characteristically high orthogonality and their average aspect ratios are close to 1 [68]. This implies that the **IDW** weighting, shown necessary for tetrahedral meshes by Mavriplis [83], is not necessary for polyhedral meshes. The only difference between the **IDWLSG**

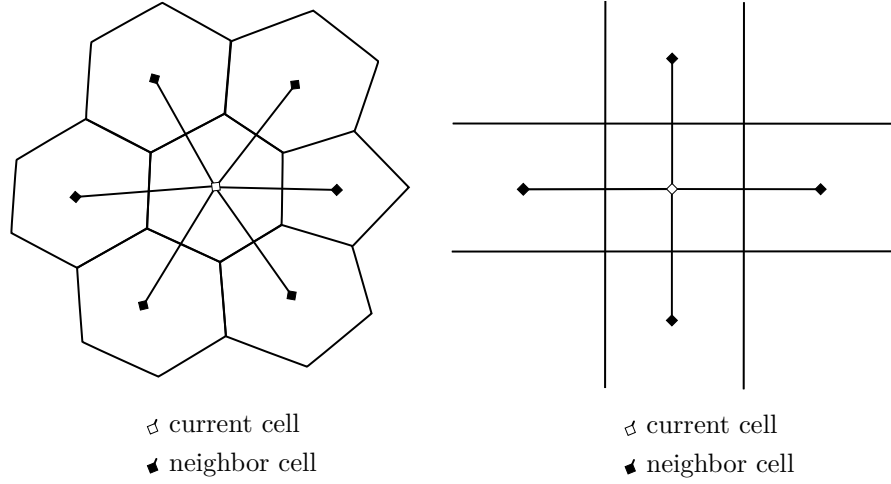


Figure 11: The number of cell faces determines size of  $\mathcal{C}$  which in turn determines the accuracy of  $\nabla_c$ . Introducing vertex-cell neighbors as proposed by Mavriplis [83] requires explicit stencil reconstruction on hexagonal and hexahedral unstructured meshes, because there exist cells in those meshes that are mutually connected only by their corner points.

and **LLSG** algorithm is the introduction of inverse distance weights in the minimized error functional

$$\epsilon_l = \sum_{n \in \mathcal{C}} \left[ w_n \left( \alpha_n - \frac{\int_{V_n} \alpha_l(\mathbf{x}) dx}{V_n} \right) \right]^2, \quad (85)$$

where  $w_n$  is the inversed distance weight

$$w_n = \frac{1}{\sum_{\tilde{n} \in \mathcal{C}} \frac{1}{\|\mathbf{x}_c - \mathbf{x}_{\tilde{n}}\|^p}}. \quad (86)$$

The weight exponent is set to  $p = 1$ , since all adjacent cells should have the same amount of influence in the gradient approximation.

Correa et al. [27] compare different gradient operator approximations on unstructured meshes in detail. Their research is aimed at accurate volume rendering in Computer Graphics (CG). Nevertheless, their findings can be directly used for the gradient approximation on unstructured meshes in order to obtain a reasonably accurate initial PLIC interface orientation. The following implications made by Correa et al. [27] have had an impact on the  $\nabla_c$  approximation in the present work:

1. Inversed distance based gradient approximations are generally more accurate, especially on unstructured meshes with non-equidistant cells.
2. Inversed distance based methods are more cost effective compared to regression based methods.
3. An increase in the discretization stencil size is important for improving the absolute accuracy of the approximated gradient on hexahedral meshes.

The conclusions by the aforementioned authors have been taken into consideration when the gradient approximation of the proposed geometrical VoF method was developed. It is important to note that the gradients reconstructed over the expanded cell stencil are not used for the discretization of differential operators, they only result with a better estimate of the orientation vector for the PLIC interface. Thus reconstructed gradients are not used outside of the single layer of cells that contain the PLIC interface.

#### 5.1.1.2 Mosso-Swartz algorithm

The derivation and numerical analysis of the Mosso-Swartz (MS) algorithm was done by Swartz [135] and the algorithmic formulation for unstructured meshes was done by Mosso et al. [88]. For an interface cell  $c$ , initial  $\mathbf{n}_c$  and  $\mathbf{p}_c$  are computed using equation (82). An interface polygon is defined as the intersection between the interface cell and the interface plane  $\Pi_c = \Pi(\mathbf{p}_c, \mathbf{n}_c)$

$$\mathcal{Q}_c = \mathbb{C}_c \cap \Pi_c = \{\mathbf{x}_q : \mathbf{x}_q \in \mathbb{C}_c \text{ and } \mathbf{x}_q \in \Pi_c\}. \quad (87)$$

and each interface polygon has a centroid defined as

$$\mathbf{x}_{\mathcal{Q}_c} = \frac{1}{|\mathcal{Q}_c|} \sum_{q=1}^{|\mathcal{Q}_c|} \mathbf{x}_q. \quad (88)$$

A set of estimated interface normal vectors is computed for each cell  $c$  using the set of interface polygons  $\mathcal{Q}_{c,i}$  from all the cells adjacent to the cell  $c$

$$\mathcal{N}_c = \{\mathbf{n}_{c,n}^e : \mathbf{n}_{c,n}^e \cdot (\mathbf{x}_{\mathcal{Q}_c} - \mathbf{x}_{\mathcal{Q}_{c,n}}) = 0\}. \quad (89)$$

The modified interface normal  $\mathbf{n}_c^m$  is obtained iteratively by a least-squares minimization of the error

$$\epsilon_{\mathbf{n}}^{MS} = \sum_{n=1}^{|\mathcal{N}_c|} (\mathbf{n}_c^m - \mathbf{n}_{c,n}^e)^2. \quad (90)$$

The initial iteration starts with  $\mathbf{n}_c^m = \mathbf{n}_c$  as defined for the Youngs algorithm by equation (82). Once the new interface normal vector  $\mathbf{n}_c^m$  is obtained, the interface is reconstructed, resulting in a new set of interface polygons  $\mathcal{Q}_c$ . In order to obtain second-order convergence, the steps defined by equations 87 to 90 are repeated four times ( $m = 1, 2, 3, 4$ ) [88].

Dyadechko and Shashkov [37] propose an arithmetic average as an alternative way to compute the modified normal

$$\mathbf{n}_c^m = \frac{\sum_{n=1}^{|\mathcal{N}_c|} \mathbf{n}_{c,n}^e}{|\mathcal{N}_c|}, \quad (91)$$

which reduces the computational effort introduced by the four outer iterations, the artificial smoothing of the interface as well as the required mesh resolution. They name this modification of the Mosso-Swartz algorithm as the *Swartz* algorithm.

#### 5.1.1.3 Conservative level contour interface reconstruction

The **CLCIR** family of algorithms was originally developed by López et al. [77]. Like the modification of the Swartz-Mosso algorithm developed by Dyadechko and Shashkov [37], the **CLCIR** algorithm relies on estimating the interface normal by performing a local average of the normal vectors from the surrounding interface polygons. The difference between the **CLCIR** and Swartz-Mosso algorithm variants lies in the way the estimated normal vectors  $\mathbf{n}_{c,n}^e$  are computed. **CLCIR** algorithms rely on an iso-contour reconstruction to compute the estimated normal vectors. To polygonize the iso-contour, cell corner points are first associated with volume fraction values by means of a volume-weighted average

$$\alpha_{1,p} = \frac{\sum_{n \in \mathcal{C}}^{|\mathcal{N}_c^p|} \alpha_{1,n} V_n}{\sum_{n \in \mathcal{C}}^{|\mathcal{N}_c^p|} V_n}, V_n = \text{volume}(\mathbf{C}_n), \mathbf{C}_n \cap \mathbf{x}_p = \mathbf{x}_p, \quad (92)$$

where  $\mathbf{x}_p$  is the  $p$ -th point in the list of mesh points  $\mathbb{P}$ . From the point values defined by equation (92), for  $\lambda \in [0, 1]$  an edge iso-contour point  $\mathbf{x}_{e,\lambda}$  is subsequently defined as

$$\mathbf{x}_{e,\lambda} = \mathbf{x}_{e,0} + s(\mathbf{x}_{e,1} - \mathbf{x}_{e,0}), \text{ where } s \in [0, 1], \alpha(\mathbf{x}_{e,\lambda}) = \alpha_\lambda \quad (93)$$

and where  $\mathbf{x}_{e,0}, \mathbf{x}_{e,1}$  are the respective first and second point of a mesh edge. The parameter  $s$  is found using root finding methods when the field used for the iso-contour reconstruction is approximated in space, e.g., using a polynomial

approximation. López et al. [77] have used a linear approximation of  $\alpha$  along the edge, which results in an explicit expression for the  $s$  parameter:

$$s = \frac{\alpha_\lambda - \alpha_0}{\alpha_1 - \alpha_0}, \quad (94)$$

where  $\alpha_{0,1} = \alpha(\mathbf{x}_{e,0,1})$ . López et al. [77] have used  $\alpha_\lambda = 0.5$  for their normal estimation algorithm. Once all the edge points with  $\alpha_\lambda = 0.5$  have been reconstructed, in each interface cell  $c$  the polygonization of the iso-contour is computed by connecting edge points  $\mathbf{x}_{e,\lambda}$  in each cell  $c$ , while maintaining the interface normal orientation by enforcing outward orientation of the face area normal vectors. The **CLCIR** polygonization restricts the number of polygons per interface cell to 1. The interface normal estimation relies on the centroid of the iso-contour polygon in each cell,

$$\mathbf{x}_{c,\lambda} = \frac{1}{N_c^{ep}} \sum_{i=1}^{N_c^{ep}} \mathbf{x}_{e,\lambda,i}, \quad (95)$$

where  $N_c^{ep}$  is the number of iso-contour edge points in the cell  $c$ . The centroid  $\mathbf{x}_{c,\lambda}$  is subsequently connected with *oriented* iso-contour polygon centroids in the neighboring cells into a set of triangles according to

$$T = \{\mathcal{T}_n : \mathcal{T}_n = (\mathbf{x}_{c,\lambda}, \mathbf{x}_{n,\lambda}, \mathbf{x}_{\Pi_c(n+1),\lambda}), n = 1, \dots, |\mathcal{C}|\}. \quad (96)$$

López et al. [77] use the cross-stencil for  $\mathcal{C}$  on structured Cartesian meshes and they report second-order error convergence. Second-order convergence is achieved by extending the triangulation to include the polygon centroids in the neighboring cells, compared to the local reconstruction proposed by Shin and Juric [125] for their Level Contour Reconstruction Method (**LCRM**) method<sup>1</sup>.

Consistent triangle orientation of the triangulation  $T$  is necessary, otherwise the interface normal is falsely approximated. For the triangulation  $T$ , consistent triangle orientation is given by

$$\nexists i, j : \mathbf{n}_{\mathcal{T},i} \cdot \mathbf{n}_{\mathcal{T},j} < 0 \text{ with } \mathcal{T}_j, \mathcal{T}_i \in T \quad (97)$$

where  $\mathbf{n}_{\mathcal{T}}$  is the triangle normal vector. This is achieved by orienting all iso-contour polygon centroids  $\mathbf{x}_{n,\lambda}$  in the surrounding cells with respect to the initial interface normal vector  $\mathbf{n}_c$ . A consistently oriented iso-contour centroid triangulation  $T$  is then used to compute the first modified interface normal vector

$$\mathbf{n}_c^m = \frac{\sum_{n \in \mathcal{C}}^{|\mathcal{T}|} \mathbf{n}_{\mathcal{T},n} \alpha_n}{\sum_{n \in \mathcal{C}}^{|\mathcal{T}|} \alpha_n}. \quad (98)$$

This approximation of the modified interface normal vector is similar to the arithmetic average proposed by Dyadechko and Shashkov [37] in their modification

<sup>1</sup> The **CLCIR** normal relates the geometrical **VoF** method with the hybrid Level Set / Front Tracking method, in the sense that an iso-surface is reconstructed and used for interface normal vector approximation.

of the **MS** algorithm. The corrected normal is used for the interface positioning sub-step of the interface reconstruction algorithm that results in the new interface polygons. López et al. [77] have proposed a repetition of the iso-contour polygonization step as well as a Bézier spline interface approximation to increase the interface orientation accuracy and convergence.

Their results [77, tables 1 and 2] do not show significant improvements in convergence and absolute accuracy by introducing a repeated polygonization step, nor by introducing the Bézier triangle patch interpolation. An increased accuracy and convergence are not to be expected for the repeated polygonization of the iso-contour, since the reconstruction of the interface does not modify the values of  $\alpha$ , and therefore has no impact to the volume weighted values given by equation (92) used to polygonize the iso-contour so the details of the Bézier interface approximation are omitted here.

The **CLCIR** method shows a promising and stable second-order of convergence across different mesh densities and is to be considered a good candidate for unstructured meshes, as well as the **MS** modification proposed by Dyadechko and Shashkov [37], since both methods disregard computationally expensive outer iteration steps.

#### 5.1.1.4 Linear least squares fit algorithm

The **LLSF** algorithm was proposed by Scardovelli and Zaleski [120] and extended to 3D by Aulisa et al. [14]. The algorithm starts by an initial estimate of the normal orientation using the Youngs' algorithm given by equation (82). So computed  $\mathbf{n}_c$  is then used to position the interface while upholding the prescribed volume fraction value  $\alpha$ . A positioned interface plane defines the interface polygon  $\mathcal{Q}_c$  with its centroid  $\mathbf{x}_{\mathcal{Q}_c}$ . Second-order convergence is obtained minimizing a plane functional built from all interface centroids  $\mathbf{x}_{\mathcal{Q}_c}$  in the surrounding cell neighborhood  $\mathcal{C}$ .

A distance between the interface polygon centroid in the current cell  $c$  and the centroid in the neighbor cell  $i$  is defined as

$$d_n = \|\mathbf{x}_{\mathcal{Q}_{c,n}} - \mathbf{x}_{\mathcal{Q}_c}\| \quad (99)$$

and it is used to compute the average distance

$$d_a = \frac{1}{|\mathcal{C}|} \sum_{n \in \mathcal{C}} d_n. \quad (100)$$

The individual and the average distance define the variance

$$\sigma^2 = \frac{1}{|\mathcal{C}|(|\mathcal{C}| - 1)} \sum_{n \in \mathcal{C}} (d_n - d_a)^2 \quad (101)$$

used to compute the individual weight of each neighboring centroid  $\mathbf{x}_{\mathcal{Q}_{c,n}}$  via

$$w_n = e^{\frac{-d_n^2}{a\sigma^2}} \quad (102)$$

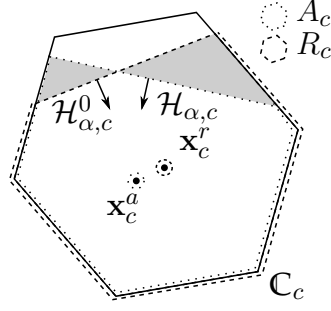


Figure 12: The reconstructed  $\mathbf{x}_c^r$  and advected  $\mathbf{x}_c^a$  phase centroid in an interface cell used by the **MoF** orientation algorithm.

with  $a$  being a free parameter set to 0.75 by the authors. The individual weight is then normalized according to

$$\tilde{w}_n = \frac{w_n}{\sum_{n \in \mathcal{C}} w_n}. \quad (103)$$

A weighted distance between a plane passing through that point with a corrected normal  $\mathbf{n}_c^m$  and each neighboring polygon centroid  $\mathbf{x}_{Q_c,n}$

$$\epsilon_{\mathbf{n}}^{LLSF} = \sum_{n \in \mathcal{C}} w_n [\mathbf{n}_c^m \cdot (\mathbf{x}_{Q_c,n} - \mathbf{x}_{Q_c})]^2, \quad (104)$$

is to be minimized, resulting in a  $3 \times 3$  linear algebraic equation system per cell  $c$  for 3 components of the corrected interface orientation vector  $\mathbf{n}_c^m$ , similar as for the **MS** algorithm. The difference between **LLSF** and the **MS** algorithms is the minimized functional. The **MS** algorithm minimizes the difference between the normal vectors and the **LLSF** algorithm minimizes the distance to a plane.

The **LLSF** method ensures a stable second-order convergence for a reconstructed sphere and its absolute accuracy is comparable to **CLCIR** [77, Table 1]. Table 1 sets the **LLSF** algorithm into a category of efficient algorithms with the computational cost that is reported to be only 1.5 times larger than the cost of the Youngs' algorithm.

#### 5.1.1.5 Moment of fluid algorithm

The **MoF** orientation algorithm Dyadechko and Shashkov [37] relies on the Youngs algorithm for the initial estimate of the interface normal. The second-order convergent improvement of the initial estimate is obtained by minimizing the distance between the *reconstructed phase centroid* and the *advected phase centroid*.

Figure 12 shows the reconstructed  $\mathbf{x}_c^r$  and the advected phase centroid  $\mathbf{x}_c^a$ . The reconstructed phase centroid is a centroid of the intersection between the positive half-space of the reconstructed interface and the corresponding cell

$$R_c = \mathcal{H}_{\alpha,c} \cap \mathbf{C}_c, \quad (105)$$



$$\mathbf{x}_c^r = \frac{1}{|R_c|} \int_{R_c} x dx. \quad (106)$$

The advected phase centroid is *initially* defined as a result of the intersection between the positive half-space given by the initial interface geometry used to set the volume fraction field and the interface cell

$$A_c = \mathcal{H}_{\alpha,c}^0 \cap \mathbf{C}_c, \quad (107)$$

$$\mathbf{x}_c^a = \frac{1}{|A_c|} \int_{A_c} x dx. \quad (108)$$

After the initial time step, the advected phase centroid is traced in a Lagrangian manner, by solving a kinematic equation for the centroid motion using the barycentric phase velocity. The initial half-space  $\mathcal{H}_{\alpha,c}^0$  in this case is used to pre-process the volume fraction field. Alternatively, the initial volume fraction value can be set by intersecting two discretized domains with each other [10]. In this case  $A_c$  is defined as an intersection between the input domain and the interface cell in the initial time step. Section 5.2 describes the MoF advection in detail. At this point, it is assumed that the advected phase centroid  $\mathbf{x}_c^a$  is available in each interface cell in the current time step.

The Youngs' reconstruction algorithm introduces an error in the orientation of the interface normal vector  $\mathbf{n}_c$ . This error is schematically shown as the shaded region in figure 12. The goal of the MoF orientation algorithm is to minimize this shaded region by modifying the direction of the normal vector  $\mathbf{n}_c$ . The difference between the advected and the reconstructed centroid,

$$\epsilon_c^{MOF} = \|\mathbf{x}_c^r - \mathbf{x}_c^a\|^2, \quad (109)$$

is minimized to compute the corrected interface normal  $\mathbf{n}_c^m$ , resulting in a new advected phase volume centroid  $\mathbf{x}_c^a$  in each optimization step. The MoF orientation algorithm [37, 6, 38] improves the reconstruction in two important ways. The reconstruction procedure is local to the interface cell, making the reconstruction algorithm massively parallel because no communication between processes that contain interface cells is required during the reconstruction step. This does not incur perfect linear scaling however, because the amount of time required by the reconstruction will be linearly proportional to the number of interface cells handled by the parallel process. When the interface cells are not uniformly distributed among parallel processes, load imbalance occurs that negatively impacts scaling. Not requiring parallel communication for the interface reconstruction is unlike all the aforementioned normal orientation algorithms. Note, however, that the reconstruction does require the phase centroid to be available and the parallel computation therefore has an additional overhead of both tracing and communicating phase centroids. Absolute accuracy of the method is much higher than for other

orientation methods, making it a better choice for problems where local dynamic refinement is required. Additionally, the centroid-based optimization results in a more accurate automatic nested reconstruction of multiple interfaces for multiphase flows [38].

### 5.1.2 Interface positioning

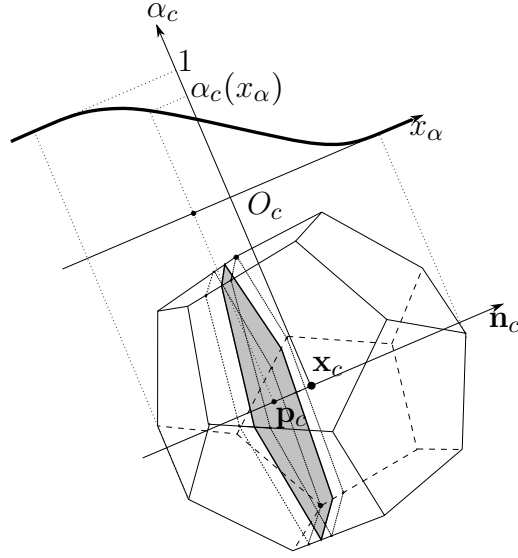


Figure 13: A schematic representation of the interface positioning. The volume fraction function  $\alpha_c(\mathbf{p}_c)$  is shown for a fixed interface normal orientation  $\mathbf{n}_c$ . Irrespective of  $\mathbf{n}_c$  and the shape of the cell, the  $\alpha_c$  function has diminishing gradients at the interval endpoints.

Whereas reconstruction algorithms differ strongly in the choice of the interface orientation algorithm, the same interface positioning algorithms are shared by many reconstruction algorithms. The aim of each positioning algorithm is to compute the position of the interface plane  $\mathbf{p}_c$ . To achieve this, a piecewise-linear approximation of the interface is used to redefine the volume fraction equation (6) for a cell  $\mathbf{C}$  as

$$\alpha_c = \frac{R_c}{V_c} = \alpha_c(\mathbf{n}_c, \mathbf{p}_c). \quad (110)$$

The interface orientation algorithm provides  $\mathbf{n}_c$ , and the  $\alpha_c$  is given either by pre-processing or by the advection step. Therefore,  $\mathbf{p}_c$  remains as the only unknown variable in equation (110). Figure 13 shows the volume fraction as a function of the interface position  $\mathbf{p}_c$  along a given orientation vector  $\mathbf{n}_c$ . It becomes obvious by inspecting figure 13 that equation (110) can be further simplified to a scalar equation

$$\alpha_c = \alpha_c(x_\alpha), \quad (111)$$

where  $x_\alpha$  is the coordinate on the  $\mathbf{n}_c$  axis with respect to an *arbitrarily chosen* origin  $O_c$ . In this thesis, the centroid of the cell  $\mathbf{x}_c$  is chosen as the origin. To compute

the interface position, the positioning algorithm reformulates equation (111) into an inverse dependency

$$x_\alpha = x_\alpha(\alpha_c). \quad (112)$$

Dyadechko and Shashkov [37] have proven that  $\mathbf{n}_c$  and  $\alpha_c$  are sufficient to compute  $x_\alpha$  (therefore also  $\mathbf{p}_c$ ) because the function given by equation (111) is a strictly monotone function. However, at the point of writing this text, there exists no single explicit function given by equation (112) that is valid for all arbitrarily shaped cells. This implies that the interface position  $x_\alpha$  needs to be determined iteratively, at least to some extent.

The actual graph of  $\alpha_c(\mathbf{n}_c, \mathbf{p}_c)$  is different for different primitive cell shapes using the same  $\mathbf{n}_c$  as shown in figure 14 for an intersected tetrahedron, cube and star polyhedron in figures 14(a) to 14(c), respectively. Regardless of the graph, the function has a diminishing derivative at two positions  $\mathbf{p}_c$ : the point where the intersection between the half-space defined by the PLIC interface plane and the cell is the complete cell, and another point where the intersection result is an empty set. The diminishing derivative causes divergence of slope-based numerical methods used to solve equation (112) for  $\mathbf{p}_c$ .

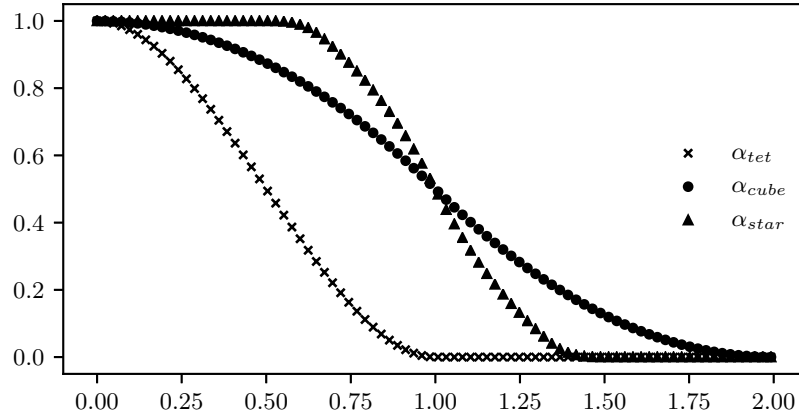
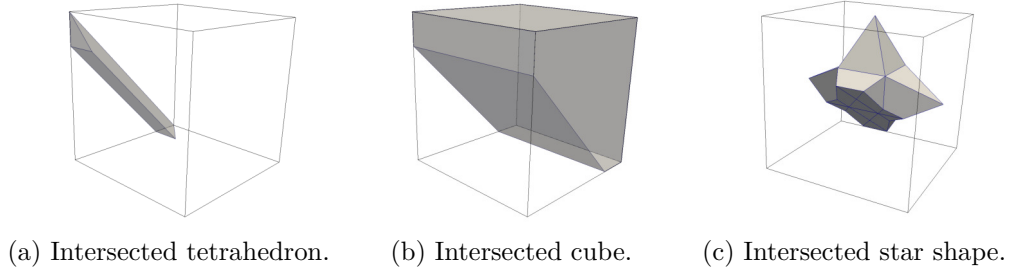


Figure 14: Volume fraction  $\alpha_c$  as a function of the interface plane position  $\mathbf{p}_c$  computed using the same  $\mathbf{n}_c = (1, 1, 0)$  for three different cell shapes.

Several interface positioning algorithms have been developed so far to compute  $x_\alpha$ . An iterative approach based on the Brent's method [21] has been used by Rider and Kothe [111]. Equation (112) is evaluated iteratively, starting with an initial guess  $x_\alpha^0$ , until  $x_\alpha$  is computed, such that the prescribed volume fraction  $\alpha_c$  is obtained up to a prescribed tolerance. The Brent method [20] is applied because

it provides a stable solution to the root finding problem even when subjected to diminishing gradients of the function at the interval endpoints as shown in figure 13 schematically and in figure 14 for actual volume fractions. Rider and Kothe [111] already state that Newton's iterative method results in lower number of iterations as well as a smart initial starting position for the Brent's method. In order to better estimate  $x_\alpha^0$  in the first step, Rider and Kothe [111] sort the cell points with respect to the projection on  $\mathbf{n}_c$

$$\mathcal{P}_{c,\Sigma} = \{\mathbf{p}_i : (\mathbf{p}_i \cdot \mathbf{n}_c) \leq (\mathbf{p}_{i+1} \cdot \mathbf{n}_c), \mathbf{p}_i \cap \mathbb{C}_c = \mathbf{p}_i\}. \quad (113)$$

To each  $\mathbf{p}_i \in \mathcal{P}_{c,\Sigma}$ , a volume fraction is assigned as

$$\alpha_{c,i} = \frac{\mathbb{C}_c \cap \mathcal{H}(\mathbf{n}_c, \mathbf{p}_i)}{V_c}, \quad (114)$$

where  $\mathcal{H}$  is a positive half-space at point  $\mathbf{p}_i$  whose orientation is defined by  $\mathbf{n}_c$ . The *cut-out slab* is defined as

$$\mathcal{S}_{c,i} = \{\mathbb{C}_c \cap \mathcal{H}(\mathbf{n}_c, \mathbf{p}_i) \cap \mathcal{H}(-\mathbf{n}_c, \mathbf{p}_{i+1}) : i = 1, 2, \dots, |\mathcal{P}_{c,\Sigma}|\}. \quad (115)$$

From equations (113) and (114), we have

$$\alpha_{c,i} > \alpha_{c,i+1}. \quad (116)$$

and as a consequence

$$\exists! \mathcal{S}_{c,i} : \alpha_{c,i+1} \leq \alpha_c < \alpha_{c,i} \iff \mathbf{p}_c \in \mathcal{S}_{c,i}. \quad (117)$$

That cut-out slab  $\mathcal{S}_{c,i}$  is then chosen which contains the interface position. At this point, Rider and Kothe [111] apply Brent's method [21] to locate the interface within the slab with lower iteration count.

A semi-analytical approach was extended to arbitrary cell shapes by López and Hernández [76] that is iterative in the fact that it relies on the sorting step and slab calculation used by Rider and Kothe [111]. Contrary to the algorithm proposed by Rider and Kothe [111], once  $\mathcal{S}_{c,i}$  is found, an analytical expression is used to compute  $\mathbf{p}_c$  explicitly. *This is possible since the slab is a convex polyhedron.* The semi-analytical approach is faster on cubic cells compared to the Brent method used in [111], no iterations are required to position the interface within the slab. Furthermore López and Hernández [76] state that the sorting and slab calculation step is still the most computationally expensive part of the positioning algorithm, even more so for cells with larger number of points.

A simplified, fully iterative and much faster approach was proposed by Ahn and Shashkov [6] that also works well with cells of arbitrary shape. Additionally, the average number of iterations for cubic cells is reported in [6] to be smaller than 8 - which is the number of vertices of the cube. This makes the stabilized iterative algorithm faster than the semi-analytical one, even for  $\Omega_h$  with hexahedral cells, since the overall number iteration count is smaller than for the semi-analytical algorithm where it is necessary to create the cut-out slabs. Moreover, the algorithm

is simpler than all other interface positioning algorithms: it consists of a combination of the secant method and the bisection method. The bisection method is used to stabilize the root finding algorithm when  $\alpha_c$  approaches 0 or 1.

Another semi-iterative interface positioning algorithm was proposed by Diot et al. [35] for planar and axis-symmetric convex cells. The authors show that their approach is faster than the standard Brent's iterative method and the algorithm proposed by Dyadechko and Shashkov [37] that relies on the calculation of the interface position  $x_\alpha$  within a slab  $\mathcal{S}_{c,i}$  using interpolation. Table 2 contains results for the axis-symmetric geometry and shows exactly what is to be expected: the average global iteration number for both the method of Dyadechko and Shashkov [37] and Diot et al. [35] are the same, since they both rely on the same sorting and slab calculation steps. The authors make no reference to Ahn and Shashkov [6], nor do they compare their results to this efficient stabilized Secant/Bisection interface positioning algorithm. Furthermore, they do not show a comparison with the method proposed by López and Hernández [76], even though the motivation for their proposed algorithm was an increased efficiency compared to this algorithm. Consequently, no conclusions can be drawn with respect to the difference in computational efficiency between the method proposed by Diot et al. [35] and the methods of López and Hernández [76] and Ahn and Shashkov [6].

Diot and François [34] have extended their 2D and axis-symmetric method [35] to 3D for convex cells of arbitrary shape. Following their work in [35], explicit analytic expressions for computing a volume of 3D slabs  $\mathcal{S}_{c,i}$  are proposed. Only a comparison with an iterative method based on Brent's root finding method is provided in the results section of the article. Accuracy and efficiency comparisons against more recent algorithms such as the algorithms by López and Hernández [76], Ahn and Shashkov [6] are not provided. Therefore, it is difficult to conclude if the 3D implementation of the volume calculation expressions actually improves the efficiency and accuracy compared to the aforementioned algorithms.

All algorithms except the one by Ahn and Shashkov [6] ([111, 37, 76, 35, 34]) rely on the pre-sorting procedure and slab calculation, which introduces *at least*  $N_p$  intersections where  $N_p$  is the number of cell points. Therefore, the positioning algorithm proposed by Ahn and Shashkov [6] is adopted in this thesis as the primary choice for interface positioning, since its average per-cell iteration count is lower than  $N_p$ .

## 5.2 VOLUME FRACTION ADVECTION

As covered in the overview of the volume fraction advection algorithm in section 4.1.3, the problems arising in the volume fraction advection are strongly related to geometrical operations performed on the flux volumes: their linearization, correction for volume conservation and subsequent intersection with interface cells and half-spaces. In this section, a chronological exposition of proposed methods that have improved on the volume fraction advection problem is given. Eulerian flux-based geometrical **VoF** methods are covered in more detail, as they are more closely related to the method proposed in the present work.

Initial developments of volume fraction advection algorithms have reduced the complexity of the geometrical operations by adopting a *dimensionally split* approach to advecting the volume fraction. This kind of advection is also called *operator splitting*. The dimensionally split advection splits the calculation of the discretized volume fraction equation (71) in  $D$  steps, where  $D$  is the spatial dimension of the solution domain. Dimensionally split algorithms update volume fraction values once per splitting step and a new geometrical approximation of the interface must be computed for the subsequent calculation step. Because the dimensionally split algorithm performs 3 interface reconstructions per time step computational cost is increased. The directional splitting still requires the calculation of the fluxed phase volume contributions across finite volume faces in equation (71). This, in turn, imposes a requirement on each finite volume face  $f$ : its normal area vector must be aligned with a coordinate axis. For structured hexahedral meshes, this requirement is fulfilled. However, unstructured meshes do not fulfill this requirement, even if the cells of the unstructured mesh are of hexahedral shape. The reason lies in the definition of the face on unstructured meshes as an ordered collection of mesh points given by equation (13). It directly follows that defining the face area normal vector with equation (22) does not imply any kind of alignment with the coordinate axes. In case of unstructured meshes with hexahedral cells no information regarding the axis alignment of face area normal vectors is used and would therefore need to be re-computed. For problems without topological mesh changes, this incurs a single calculation of the axis alignment information at the beginning of the simulation. However, for problems that require topological mesh changes, the information on the axis alignment needs to be computed after every respective topological mesh change. The aforementioned reasons and an improved overall solution accuracy [69, 70, 73] make the dimensionally unsplit advection algorithms a preferred choice for unstructured meshes. A detailed review of dimensionally split algorithms on structured meshes is given by Tryggvason, Scardovelli, and Zaleski [139] and the reader is directed to the works of Kothe et al. [69], Rider and Kothe [111], Scardovelli and Zaleski [119, 120], Aulisa et al. [14] for details on individual algorithms.

The differences between the state of the art dimensionally unsplit algorithms are in the algorithm sub-tasks: geometrical approximation of the flux volume, corrections for volume conservation and the geometrical intersection operations required for computing the fluxed phase volume contributions.

Kothe et al. [69] and later Rider and Kothe [111] proposed a two-dimensional Eulerian flux-based unsplit algorithm that uses a constant velocity distribution across an edge (face in  $3D$ )  $\mathbf{u}_f$  to construct the flux volume  $V_f$  and consequently the fluxed phase volume contributions  $V_{f,c}^\alpha$ . However, they noted that Pilliod and Puckett [103] were the first to develop a directionally unsplit multidimensional algorithm ([69, page 3, table 1], [111, page 6, table 1]). The computation of the phase volume contributions given by equation (69) by the Rider-Kothe Algorithm (RKA) is shown schematically in figure 15. Compared to using point velocities as shown in figure 6, the use of constant velocities by the RKA causes an overlap between the flux volumes and subsequently the phase flux contributions for two point-adjacent edges, i.e. faces. The overlap is *fluxed twice* and shown schematically as the shaded

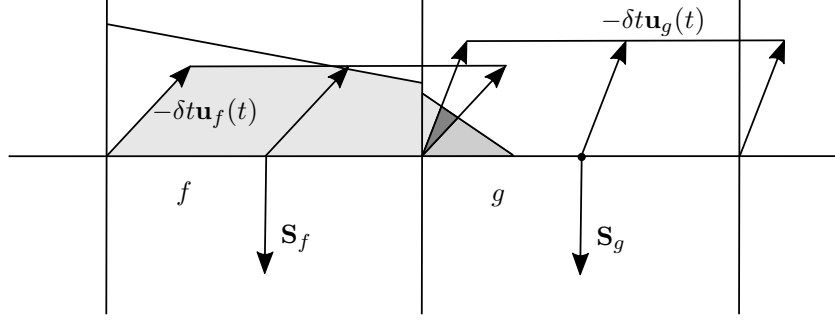


Figure 15: A schematic representation of the 2D Rider-Kothe algorithm [111].

triangle in figure 15. Fluxing the same volume multiple times this way causes *overshoots* and *undershoots*. An overshoot is defined as

$$\alpha_c^o = \{\alpha_c - 1 : \alpha_c > 1\} \quad (118)$$

and an undershoot is defined as

$$\alpha_c^u = \{-\alpha_c : \alpha_c < 0\}. \quad (119)$$

Undershoots and overshoots have been handled in the **RKA** using explicit conservative redistribution of the volume fraction  $\alpha$ . A second-order convergent reconstruction algorithm **ELVIRA** [103] has been used to reconstruct the **PLIC** interface. Since Rider and Kothe [111] have used edge (face) centered velocities to construct the flux volumes, there is no need to correct the geometrical flux volumes for volume conservation. However, this is only true if the edge-centered velocity field upholds the discrete divergence free condition  $\sum_f F_f = 0$ . Rider and Kothe [111] do propose a correction for volume conservation, but for different reasons: they expect the face centered velocity field to uphold the discrete divergence free condition *up to a specified tolerance of a linear solver used to obtain it*. They were aiming at applying their algorithm with velocity fields that result from an approximated solution of the incompressible single field Navier-Stokes (**NS**) equation system. In that case, a following correction is proposed:

$$\partial_t \alpha + \nabla \cdot (\mathbf{u} \alpha) = \alpha \nabla \cdot \mathbf{u}. \quad (120)$$

However, should an explicit function that adheres to the discrete divergence free criterion be used to evaluate  $\mathbf{u}_f$ , no correction for volume conservation is required for the geometrical flux volumes of the **RKA**. Using an edge (face) centered velocity simplifies the geometrical form of the geometrical flux volume: the flux polygon is either convex or weakly non-convex. However the velocity field varies over the edge (face), the **RKA** assumes *a constant* flux velocity over the edge (face), so the only non-convexity of the resulting flux polyhedron can be introduced by the non-convexity of the face of the control volume itself, which in turn is a result of the mesh generation algorithm. Consequently, the geometrical operations used subsequently to compute the phase volume contributions are substantially



simplified, especially in three dimensions. The authors proposed two variants of their algorithm: a fully dimensionally unsplit algorithm, and the unsplit variant constructed from the dimensionally split algorithm. In the latter algorithm, a corner flux polygon correction procedure is required to cut out large corner overlaps characteristic for the dimensionally split algorithms. In both cases the cell corner velocities are determined from the edge center velocities based on their signs. Rider and Kothe [111] have shown that their unsplit algorithm is more accurate than the operator split variant.

Mosso et al. [89] were the first to propose a cell-based (re-mapping) **LE** method for the dimensionally unsplit volume fraction advection that uses a forward projection. They reported a test case involving a translation of a circle on an unstructured irregular hexahedral mesh using a periodical spatially constant velocity field, that moves the circle back to the original position. The algorithm shows promising results [89, figure 5] in the fact that the shape and the area of the circle are maintained even on a non-orthogonal unstructured hexahedral mesh.

Mosso et al. [88] described the application of their **LE** method to the problem of a rotating planar and circular interface and the numerical errors that are related to exact, forward Euler, backward Euler and trapezoidal integration of mesh point displacements. They concluded that the use of the trapezoidal integration for the forward projection step of the **LE** method removes artificial expansion and contraction of the interface. In other words, the trapezoidal integration of point displacements conserves volume - a conclusion that is also drawn later by Le Chenadec and Pitsch [71].

In their comprehensive review of methods for **DNS** of two-phase flows, Scardovelli and Zaleski [119] stated that the dimensionally unsplit algorithm provide better accuracy compared to their split counterparts, especially regarding asymmetries in the shape of the advected interface.

Harvie and Fletcher [53] have proposed the *stream scheme*: a two-dimensional Eulerian flux-based geometrical **VoF** method that uses a continuous velocity field approximation and a geometrically reconstructed interface to compute the fluxed phase volume contributions by approximating the flux volume  $V_f$  as a set of *stream tubes*. The velocity field is based on a streamline function formulated as a bilinear polynome

$$\Psi(x, y) = (\chi_b x + \chi_y)\mathbf{i} - (\chi_b y - \chi_x)\mathbf{j}. \quad (121)$$

The velocity is given directly from the stream function as

$$\mathbf{u} = \begin{pmatrix} \partial_y \Psi \\ -\partial_x \Psi \end{pmatrix} = \begin{pmatrix} \chi_b x + \chi_y \\ -\chi_b y - \chi_x \end{pmatrix}. \quad (122)$$

The divergence free condition  $\nabla \cdot \mathbf{u} = 0$ , when applied to the velocity field computed from the stream function shows

$$\nabla \cdot \mathbf{u} = \partial_x(\chi_b x + \chi_y) + \partial_y(-\chi_b y - \chi_x) = \chi_b - \chi_b = 0, \quad (123)$$



which means that the volume conservation of the approximated velocity field will be upheld irrespective of the chosen coefficients of the bilinear stream function polynomial. The coefficients chosen by Harvie and Fletcher [53] are specific to Cartesian equidistant meshes and are hence not covered here. The authors note that the velocity field is continuous in the normal direction across cell faces, and discontinuous in the tangential direction, as well as at cell corner points. Once the velocity field has been approximated by the stream function, fluxed phase volume contributions are calculated based on stream tubes given by the velocity field and a discretization of the face in  $N_{stream}$  segments, as shown in figure 16. A stream tube is thus defined by the streamline of the fluid particle that crosses a face along the stream coordinate  $l$  and the width  $w$ . The stream tube width is given by  $N_{stream}$  and the velocity field  $\mathbf{u}$  from the volume conservation law [53]. The actual geometry of the stream tubes is not explicitly approximated. Instead, the fluxed phase volume contributions are calculated as integrals of the phase indicator function along the streamline. The connection with other geometrical VoF methods is the PLIC based geometrical interface approximation that substitutes the discontinuous phase indicator function.

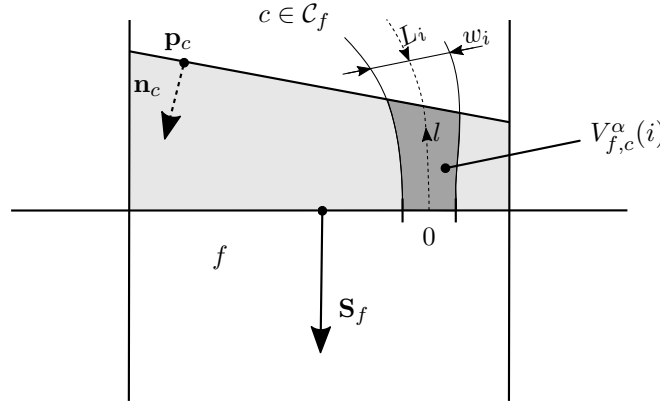


Figure 16: A schematic representation of the 2D stream scheme [53].

The fluxed phase volume contribution over the face  $f$  as

$$V_{f,c}^{\alpha} = \sum_i^{N_{stream}} V_{f,c}^{\alpha}(i) = \sum_i^{N_{stream}} \int_0^{L_i} w_i(l) I(l) dl, \quad (124)$$

where  $I$  is the phase indicator function. The stream scheme approximates the phase indicator function using a PLIC interface. Details on how the above integral is reformulated for actual calculation are available in [53]. The stream scheme has comparative results to those of Rider and Kothe [111] for the reversed single vortex test case. The authors have noted that the second-order reconstruction algorithm ELVIRA increases the accuracy of the advection. Using the first-order convergent Youngs' reconstruction algorithm in the regions where the fluid interface forms thin filaments is error prone, since the error of the gradient operator introduces instabilities in the motion of the interface that are then amplified and cause the interface to break up. The accuracy and computational cost of the stream scheme is highly dependent on  $N_{stream}$ . The authors have reported that  $N_{stream} = 10$  makes

the computational efforts comparable to other dimensionally unsplit algorithms. Numerical tests show that the scheme suffers from *wisps*: small values in  $\alpha_c$  in cells that should be empty and values near 1 in cells that should be completely filled. Wisps have orders of magnitude lower values than jetsam and flotsam first described by Noh and Woodward [95] and later by Rider and Kothe [111]. Harvie and Fletcher [53] deal with wisps by applying a conservative wisp removal algorithm that takes into account the direction of the interface normal vector. The conservative wisp removal algorithm removes wisps within the 27 cell stencil of a target cell, making it the only point in the stream scheme where the dependence on the CFL condition is introduced.

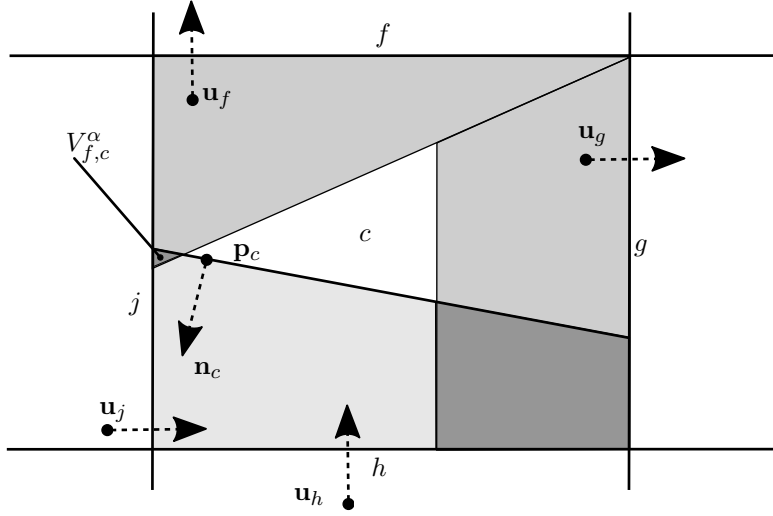


Figure 17: A schematic representation of the two-dimensional DDR scheme [52].

Harvie and Fletcher [52] have proposed the two-dimensional DDR scheme: a scheme that is inherently volume conservative and that does not produce flotsam or jetsam as well as volume fraction over-or-undershoots. The DDR scheme constructs *defined donating regions* for all faces of a cell  $c$  whose velocities are directed outwards from the cell. A defined donating region is an approximation of the flux volume  $V_f$ . Once the donating regions are constructed, as shown schematically in figure 17, intersections with the PLIC interface result in the computation of the fluxed phase volume contribution  $V_{f,c}^\alpha$  for each face  $f$ . Faces  $f, g$  and  $h$  are labeled in figure 17 to distinguish their respective velocities used for constructing the defined donating regions. The DDR scheme improves volume conservation compared to Rider and Kothe [111], Harvie and Fletcher [53] by using unique slopes for the donating regions at cell corners. The scheme is introduced in two-dimensions, and the slopes are defined as the ratio

$$\frac{dy}{dx}|_{fg} = \frac{\|\mathbf{u}_f\|}{\|\mathbf{u}_g\|}, \quad (125)$$

whereas the velocity  $\mathbf{u}_h$  is an *inflow velocity*, so no *donating region* is constructed for this face. The volume of the donating region is a result of the *total* volume conservation for the cell  $c$  and it defines the geometry of the donating region. The DDR scheme bases its calculations on Cartesian equidistant meshes and the reader

is directed to [52] for more details on the actual computation. The conservation of the total volume for the cell  $c$  not only defines the geometry of the donating regions, it also indirectly introduces the CFL criterion into the scheme. Preventing the characteristic overlap of the flux volumes in the RKA (figure 15) improves the volume conservation of the DDR scheme together with the correction of the donating region for volume conservation. However, limiting the donating regions to a cell prevents the scheme from *fluxing around the corner*.

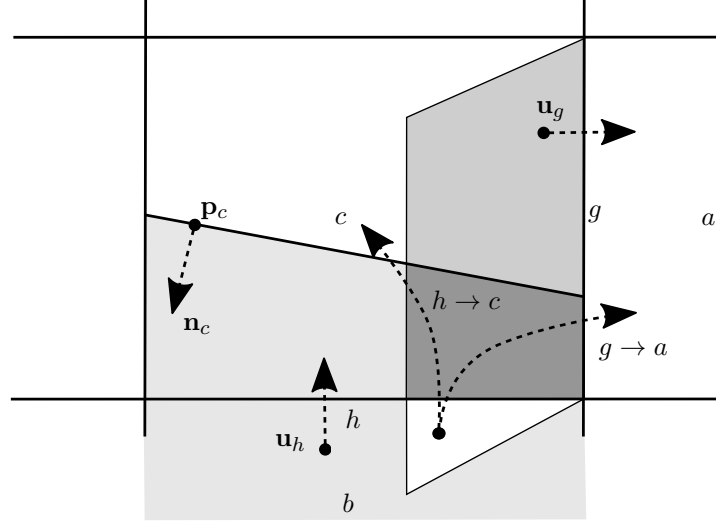


Figure 18: A schematic representation of the corner-flux ignored by the DDR scheme.

Consider figure 18 and assume for simplicity that the cell  $b$  is full with  $\alpha_c = 1$ . The white corner flux is not fluxed from cell  $b$  to cell  $a$  by the stream scheme. The donating region constructed for the donating face  $h$  of cell  $b$  will flux the triangle into cell  $c$  ( $h \rightarrow c$ ). However, for the cell  $c$ , the donating region of the outflow face  $g$  stops at the face  $h$ , since  $h$  is an inflow face for cell  $c$ . The RKA and other dimensionally unsplit algorithms that work with cell corner velocities make sure that the flux corresponding to the face  $g$  transfers the white triangle to cell  $a$  ( $g \rightarrow a$ ). Ignoring the corner fluxes is the cause of the method's larger advection errors for test cases where the velocities are directed along cell diagonals which is characteristic for: the diagonal translation case [52, table 1], slotted disk test case [52, table 3] and during the reversed vortex case [52, table 4]. The DDR scheme is outperformed by the original Youngs' scheme for all those test cases. Nevertheless, the DDR scheme is the first scheme that proposed the use of unique cell corner velocities to build discrete Lagrangian trajectories required to approximate the flux volume  $V_f$ . As a result, its volume conservation and numerical boundedness errors were exceptionally small. Additionally, enforcing the conservation of total volume removed the need for explicit flux removal or under/overshoot redistribution algorithms.

Cerne et al. [24] have analyzed the numerical errors of the geometrical VoF method. They have quantified the reconstruction errors for two-dimensional simulations on structured meshes in the form of *reconstruction correctness*. They have also described the effect of the advection errors on the distortion in the shape of the circular interface that is subjected to translation on coarse meshes. Moreover, they

show artificially high interface advection velocities for dispersed interfaces, where the length of the interface elements approaches cell size. As a solution for the interfacial dispersion, they propose the use of local dynamic **AMR** as a means of dealing with dispersion errors, based on the reconstruction correctness criterion.

Scardovelli and Zaleski [120] have formalized the geometrical interpretation of their two-dimensional Eulerian Implicit - Lagrangian Explicit (**EI-LE**), proposed originally by Aulisa, Manservigi, Scardovelli, and Zaleski [15]. The scheme formalization is based on modeling the steps of the **EI-LE** scheme using linear mappings. Scardovelli and Zaleski [120] extend the original **EI-LE** scheme as a dimensionally unsplit scheme. However, the unsplit nature of the scheme is interpreted in the **VoF** context: a single advection and reconstruction step are executed per time step. The scheme still requires the face area normal vectors to be aligned with coordinate axis. Additionally, entire scheme derivation is prepared for structured meshes. The **EI-LE** is the first scheme that reports the absence of wisps and conserves the total area exactly. A result that deserves special attention is the reversed vortex test case [15, figure 10] with no visible wisps in the solution and a reduced geometrical advection error compared to previous methods, including the **DDR** scheme. The extension of the **EI-LE** method to unstructured meshes has been suggested by the authors, but it requires a triangulation of the domain and a continuous area preserving linear mapping. Even though **EI-LE** shows very promising results, the order of accuracy of the unstructured **FVM** deteriorates to first-order of accuracy on unstructured tetrahedral/triangular meshes [61, 68]. Therefore, an extension of **EI-LE** to unstructured meshes has not been considered.

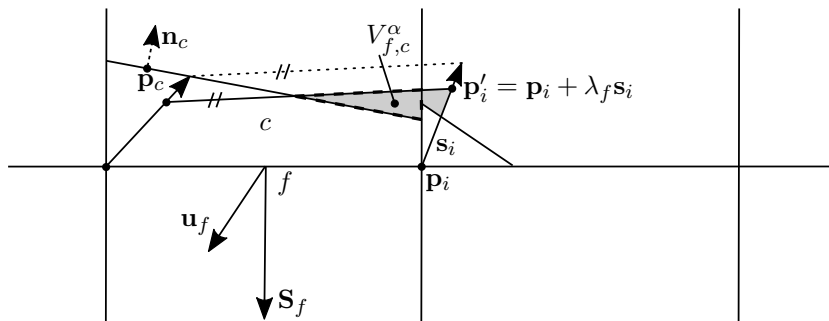


Figure 19: A schematic representation of the EMFPA scheme.

López et al. [75] have combined the volume conservation and numerical boundedness of the solution produced by the **DDR** scheme of Harvie and Fletcher [53] and the dimensionally unsplit approach proposed by Rider and Kothe [111] that does not disregard computing the around the corner fluxes into a new Edge-Matched Flux Polygon Advection (**EMFPA**) scheme. Shown schematically in figure 19, the flux volume  $V_f$  is defined by the positions of the swept face points  $\mathbf{p}'_i$  computed as

$$\mathbf{p}'_i = \mathbf{p}_i + \lambda_f \mathbf{s}_i, \quad (126)$$

where  $\lambda_f$  is a face-constant scalar coefficient and  $\mathbf{s}_i$  is the displacement along the discrete Lagrangian trajectory, computed using the **IDW** interpolation

$$\mathbf{s}_i = \frac{1}{\sum_{f=1}^{N_{pf}} w_{i,f}} \sum_{f=1}^{N_{pf}} w_{i,f} \mathbf{u}_f^{n+\frac{1}{2}} : \mathbf{p}_i \in \mathbb{F}_f, \quad (127)$$

where

$$w_{i,f} = \frac{1}{\|\mathbf{p}_i - \mathbf{x}_f\|}. \quad (128)$$

The face velocity  $\mathbf{u}_f$  that contributes to the slope is evaluated in an *intermediate time step*  $n + \frac{1}{2} \equiv t + 0.5\delta t$ : the velocity field at this time is obtained from a linear interpolation between the current and the next time step,  $n$  and  $n + 1$  respectively. Using the **IDW** interpolation to compute the slope  $\mathbf{s}_i$  of the discrete Lagrangian trajectories introduces an interpolation error. On Cartesian equidistant meshes used by López et al. [75], equation (127) represents an arithmetic average, since all the distances from corner points to face centers are the same. The slope of the discrete Lagrangian trajectory influences the overall scheme accuracy, as shown by more recently proposed dimensionally unsplit geometrical advection schemes where interpolations of higher-order are used to approximate the slope. The slope  $\mathbf{s}_i$  and the face-constant parameter  $\lambda_f$  determine the magnitude of the geometrical flux volume  $V_f$  shown as a lightly shaded polygon in figure 19. To ensure a second-order accuracy of the temporal integration for the volume fraction, the first-order accurate Euler scheme used in chapter 4 for the flux volume in equation (61) is replaced by a trapezoidal quadrature rule for the flux volume  $V_f$ ,

$$V_f = \int_t^{t+\delta t} \int_{S_f} \mathbf{u}(t) \cdot \mathbf{n} do \stackrel{S_f \neq S_f(t)}{\approx} 0.5\delta t (\mathbf{u}_f^n + \mathbf{u}_f^{n+1}) \cdot \mathbf{S}_f. \quad (129)$$

Subsequently, the geometrical flux volume is made equal to the second-order accurate flux volume  $V_f$  to ensure volume conservation, by adjusting the face-constant parameter  $\lambda_f$ . Calculation of the fluxed phase volume contributions  $V_{f,c}^\alpha$  by clipping the corrected geometrical flux volume with surrounding cells  $c \in \mathcal{C}_f$  and interface half-spaces  $\mathcal{H}_c$ . It is important to note that the second-order temporal accuracy of the **EMFPA** algorithm is ensured by the trapezoidal rule integration of the flux volume as well as the slope calculation with the intermediate velocity  $\mathbf{u}_p^{n+\frac{1}{2}}$  interpolated linearly from  $\mathbf{u}_p^n$  and  $\mathbf{u}_p^{n+1}$ . This is possible only when an explicit velocity field  $\mathbf{u} = \mathbf{u}(\mathbf{x}, t)$  is given that can be evaluated at any point in time. For simulating two-phase flows,  $\mathbf{u}_f^{n+1}$  that adheres to a discrete divergence free condition up to the tolerance of the linear solver is an approximated solution of a single-field two-phase **NS** equation system.

Volume conservation is improved by the **EMFPA** algorithm proposed by López et al. [75], compared to the original **RKA** algorithm proposed by Rider and Kothe [111] because of the reduction of the overlap between neighboring flux volumes. However, using the face-constant volume conservation adjustment coefficient  $\lambda_f$  results in possible overshoots and undershoots in the solution. Consider the cell  $c$

shown in figure 19. The magnitude of the fluxed phase volume contribution  $V_{f,c}^\alpha$  is uniquely defined by points  $\mathbf{p}_i$  and  $\mathbf{p}'_i$  as well as the interface half-space  $\mathcal{H}_c$ . The  $\lambda_f$  magnitude determines the position of the swept points  $\mathbf{p}'_i$  and it depends on the difference between the geometrical flux volume and the flux volume. This difference is a function of the slope  $\mathbf{s}_i$ : larger error in the slope of the discrete Lagrangian trajectory results in a larger correction for volume conservation. The fluxed phase volume contribution  $V_{f,c}^\alpha$  shown in dashed can be over or underestimated. As a consequence, the sum of the total volume of both phases entering cell  $c$  may be different from the sum of the total volume of both phases exiting cell  $c$ . In other words, overshoots or undershoots in the volume fraction  $\alpha_c$  can occur in the cell  $c$ . López et al. [75] show that their EMFPA algorithm coupled with the Spline Interface Reconstruction (SIR) algorithm provides an overall second-order convergence. They also emphasize the necessity for a second-order convergent reconstruction algorithm because of its strong influence in the absolute error magnitude. They do not quantify errors in volume conservation and numerical stability. The authors mention that some overshoots appear, but only in cell where the slopes  $\mathbf{s}_i$  are almost orthogonal to  $\mathbf{S}_f$ , in which case a local conservative redistribution algorithm is applied. Additionally, they state that wisps appear in the solution and that they do not have an effect on the computational efficiency of the algorithm. The effect of wisps on numerical stability is not addressed.

A dimensionally unsplit algorithm is proposed by Pilliod and Puckett [103] that is based on the work of Bell, Dawson, and Shubin [18] and relies on the method of characteristics to integrate the flux volumes in time with either first or second-order accuracy. The scheme is constructed in two-dimensions and a face-constant velocity is used for the calculation of the characteristic lines. Unlike the DDR scheme and similar to the RKA this method allows the calculation of the corner fluxes, which makes it more accurate. Pilliod and Puckett [103] show near machine tolerance volume conservation errors for a translation of a circle and the rotation slotted disc by Zalesak [149] and state a necessity for the second-order accurate VoF scheme to utilise a second-order accurate reconstruction algorithm. Verification cases involving both spatially and temporally varying velocity fields are not presented, as well as the extension to three dimensions.

Dyadechko and Shashkov [37, section 4.1] rely on the Lagrangian tracing / Eulerian remapping (LE) geometrical VoF method for advecting the volume fraction field in the framework of their proposed Moment of Fluid (MoF) method. Backward tracing of the points is used to compute the  $\alpha_c^{n+1}$  as described in section 4.2, based on the 4th-order Runge-Kutta (RK) scheme. The method was implemented in 2D, and the authors propose the use of *bins* (Axis-Aligned Bounding Box (AABB)) that simplify the geometry of polygonal cells and allow for a substantial computational cost reduction when computing contributions to  $\alpha_c^{n+1}$ . Because the Lagrangian backward tracing does not preserve the constant volume of the original cell, Dyadechko and Shashkov [37] rely on a local conservative redistribution for correcting values of  $\alpha_c^{n+1}$  that are out of bounds  $[0, 1]$ , as well as for correcting artificial interface cells that are registered as mixed, but should not be (so-called *wisp cells*).

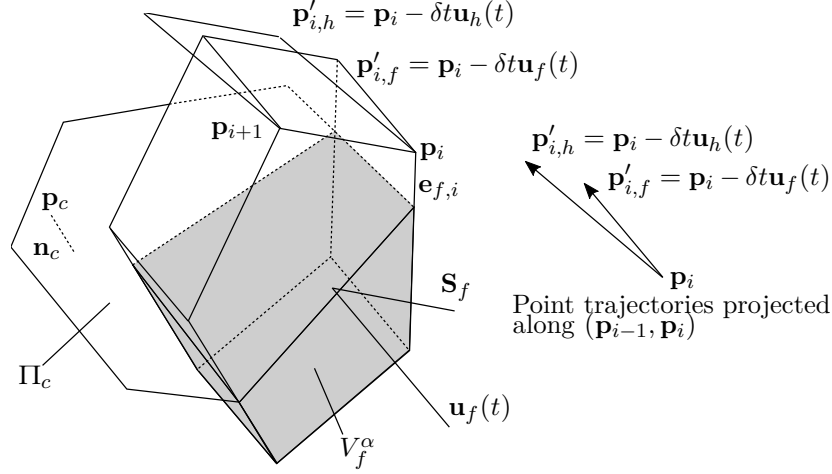


Figure 20: A schematic representation of the PCFSC advection scheme.

Piecewise Constant Flux Surface Calculation (**PCFSC**), a three-dimensional extension of the **RKA** of Rider and Kothe [111] is proposed by Liovic et al. [73]. The **PCFSC** algorithm is described for equidistant Cartesian meshes, but like **RKA**, it directly generalizes to unstructured meshes with arbitrary cell shapes. Figure 20 contains a schematical representation of the elements required by the **PCFSC** algorithm. The dimensionally unsplit advection is achieved by Eulerian flux-based backward tracing approach, as described in section 4.1 with an important simplification: the discrete Lagrangian point trajectories are not determined using unique velocity  $\mathbf{u}_p(t)$ , a face centered velocity  $\mathbf{u}_f$  is used instead. A direct consequence of using the face centered velocity  $\mathbf{u}_f$  to compute the same discrete Lagrangian trajectory for each point  $\mathbf{p}$  are planar ruled surfaces. This in turn means that no further linearization of the flux polyhedron is required. Furthermore, provided that  $\mathbf{u}_f$  satisfies the discrete divergence free condition, no correction of the flux volumes are required for volume conservation. Planar ruled surfaces and a constant point velocity result in a significant simplification of the geometrical operations required to compute the fluxed phase volume contributions  $V_{f,c}^\alpha$ , compared to the proposed method outlined in chapter 6. With the additional absence of the flux volume correction for volume conservation, **PCFSC** is computationally less expensive than other 3D schemes. However, as shown in figure 20, using face centered velocities to sweep points results in non-unique discrete Lagrangian trajectories used for a point  $\mathbf{p}_i$ . As a consequence, flux volumes may overlap or have holes between them *along the whole length of an edge*, which in turn results in fluxed phase volume contributions not to be fluxed, or to be fluxed twice, leading to overshoots and undershoots in  $\alpha_c$ . The projection of the discrete Lagrangian point trajectories in figure 20 shows how the application of a face-constant velocity for point displacements may create a hole between two edge-adjacent geometrical flux volumes. This can be observed for the edge  $\mathbf{e}_{f,i-1} = (\mathbf{p}_{i-1}, \mathbf{p}_i)$  of the face  $f$  in figure 21. For velocity fields that vary spatially across a face  $f$  a piecewise constant veapproximation of the velocity results in loss of accuracy. In an effort to efficiently



suppress flux volume errors, Liovic et al. [73] scale the fluxed phase volume with a scalar coefficient

$$\tilde{V}_f^\alpha = \Lambda_f V_f^\alpha, \quad (130)$$

where  $\Lambda_f$  is given as

$$\Lambda_f = 1 - \frac{\|\mathbf{u}_f \cdot \mathbf{S}_f \delta t\| - V_f^g}{V_f^g}, \quad (131)$$

where  $V_f^g$  is the geometrical flux volume. Liovic et al. [73] show that their **PCFSC** coupled with their **CVTNA** algorithm results in an overall second-order convergent solution for standard interface advection verification cases. Volume conservation and numerical stability errors for spatially and temporally varying velocity fields are not discussed.

Aulisa et al. [14] have extended their **EI-LE** scheme [15, 120] to support three-dimensional computations on Cartesian meshes. The dimensionally split Eulerian Implicit - Lagrangian Explicit 3D (**EILE-3D**) and Eulerian Implicit - Lagrangian Explicit 3D Decomposition Simplified (**EILE-3DS**) schemes conserve mass exactly for sphere translation and rotation test cases. **EILE-3DS** results in an asymptotical second-order convergence of the advection errors for their non-uniform vorticity single vortex test case with a decreasing **CFL** number. Aulisa et al. [14] have quantified volume conservation errors for spatially and temporally varying velocity fields. They show that the volume conservation errors of the **EILE-3DS** method are converging from approximately  $1e - 03$  to  $1e - 06$  for their single vortex test case with increased mesh resolution and from approximately  $1e - 04$  to  $1e - 09$  for the same test case with  $32^3$  volumes and a decreasing **CFL** number.

Hernández et al. [54] have proposed the Face-Matched Flux Polyhedron Advection (**FMFPA-3D**) scheme as an extension of their **EMFPA** [75] that supports three-dimensional calculations. They note that the flux integration by trapezoidal rule in equation (129) is replaced by an integration that uses intermediate face-centered velocity values when the scheme is coupled with the single field formulation of the **NS** system.

Figure 21 schematically describes geometrical flux volume construction for the **FMFPA-3D** advection scheme. As its name states, the goal of the scheme is to maintain a planar polyhedral geometrical representation of the geometrical flux volume. Hernández et al. [54] state that the three-dimensional extension that naturally follows from their **EMFPA** scheme would utilise unique point velocities, however, such extension would involve geometrical intersection operations on complex non-convex objects with non-planar ruled surfaces, which was left as future work. Instead, the **FMFPA-3D** scheme goes one step further than the **PCFSC** scheme and applies velocities that are *unique for each mesh edge*. The edge velocities are interpolated from face-centered velocities from those faces that connect to the edge. This ensures the planarity of the ruled surfaces, since all edge points are swept with the same displacement vector. However, the velocities are not unique at mesh points. Consider the point  $\mathbf{p}_i$  in figure 21: different swept points  $\mathbf{p}'_i$  are



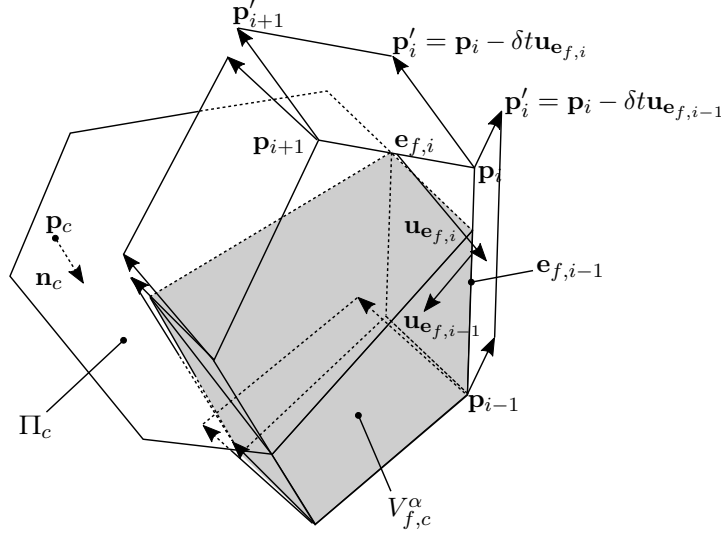


Figure 21: A schematic representation of the initial FMFPA-3D geometrical flux volume.

generated with different edge velocities  $\mathbf{u}_{ef,i}$  and  $\mathbf{u}_{ef,i-1}$ . Constructing a geometrical flux volume for the face  $f$  using non-unique point velocities for spatially varying velocity fields can leave the faces of the geometrical flux volume disconnected in the neighborhood of that point: a hole can be generated between two swept edges. The **FMFPA-3D** scheme further simplifies the geometrical flux volume by closing the gaps between swept edges by the following approximation:

$$V_f^g = \{\mathbf{x} : (\mathbf{x} - \mathbf{p}_i) \cdot \mathbf{n}_i \leq 0 : i = 1, 2, \dots, |\mathbb{F}_f| + 1\}, \quad (132)$$

where

$$\mathbf{n}_i = \begin{cases} \mathbf{u}_{e_i} \times \mathbf{e}_i, & 1 \leq i \leq |\mathbb{F}_f|, \\ \mathbf{n}_{f',l}, & i = |\mathbb{F}_f| + 1 \end{cases} \quad (133)$$

and  $\mathbf{n}_{f',l}$  is the normal vector of the linearized swept face  $f'$ . If the face  $\mathbb{F}_f$  is defined by equation (13) as an ordered (oriented) set of points, sweeping the face  $\mathbb{F}_f$  using edge velocities defines a swept face  $\mathbb{F}'_f$  as a set of all swept points  $\mathbf{p}'_i$  shown in figure 21. Using non-unique point velocities increases the number of swept points. In the case of the **FMFPA-3D** scheme, the number of swept face points doubles. Non-uniqueness of the edge velocities  $\mathbf{u}_{ef,i}$  shown in figure 21 may lead to the swept face  $\mathbb{F}'_f$  to be non-planar. In this case, the **FMFPA-3D** scheme linearizes the swept face in the form of a plane  $\Pi(\mathbf{n}_{f',l}, \mathbf{p}_{f',l})$ . The  $\mathbf{n}_{f',l}$  is defined by Hernández et al. [54] for structured Cartesian meshes. For unstructured meshes with arbitrary cell shape it can be defined as an area weighted triangle normal average,

$$\mathbf{n}_{f',l} = \frac{1}{\sum_{i=1}^{|\mathbb{F}'_f|} \|\mathbf{n}_{\mathcal{T},i}\|} \sum_{i=1}^{|\mathbb{F}'_f|} \mathbf{n}_{\mathcal{T},i}, \quad (134)$$

where the triangle normal is defined as

$$\mathbf{n}_{\mathcal{T},i} = (\mathbf{p}'_i - \mathbf{x}_{f'}) \times (\mathbf{p}'_{i+1} - \mathbf{x}_{f'}) \quad (135)$$

and  $\mathbf{x}_{f'}$  is the centroid of the swept face  $\mathbb{F}'_f$  defined by equation (20). Alternative means of computing a normal vector of a non-planar set of points are also applicable. Having computed  $\mathbf{n}_{f',l}$ , to linearize the geometrical flux volume using equation (132), the point  $\mathbf{p}_i \equiv \mathbf{p}_{f',l} \iff i = |\mathbb{F}_f| + 1$  needs to be computed. Hernández et al. [54] determine the position of this point using an explicit exact equation for the volume of a *convex polyhedron with faces as planar convex polygons* that is then used to position the point  $\mathbf{p}_{f',l}$  so that volume conservation is upheld. This computation is possible, since the geometrical flux volume simplification given by equations (132) to (135) simplify the original geometrical flux shown in figure 21 to a convex polyhedron with planar convex faces. It is necessary because the geometrical flux volume does not correspond to the flux volume given by  $\mathbf{u}_f$ .

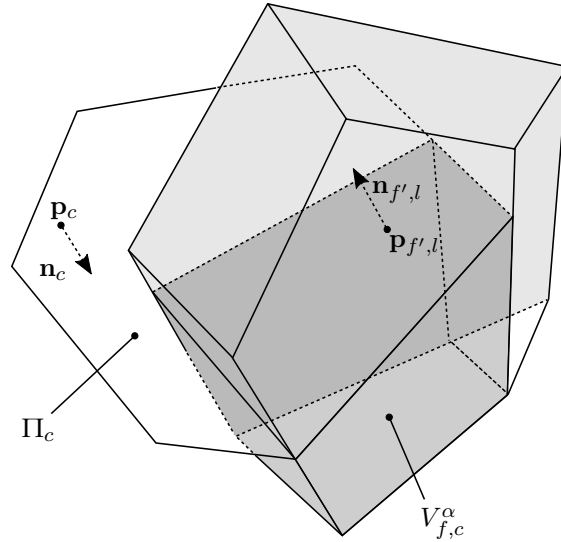


Figure 22: A schematic representation of the linearized FMFPA-3D geometrical flux volume.

Figure 22 shows the final linearized geometrical flux volume, corrected for volume conservation. The non-unique point displacements are replaced by plane intersections given by inequalities in equation (132). Note that the non-planar swept face  $\mathbb{F}'_f$  is linearized using the plane  $\Pi(\mathbf{n}_{f',l}, \mathbf{p}_{f',l})$ .

Hernández et al. [54] have compared their results to their 3D implementation of the RKA, where they have adjusted the swept face position approximately for volume conservation. It is not clear why the adjustment of the face position in the RKA implementation is done approximately, because an exact correction is used for the swept face in the FMFPA-3D algorithm with a more complex flux linearization compared to the RKA.

Zhang and Liu [154] have proposed the 2D Polygonal Area Mapping Method (PAM), a 2D LE geometrical VoF method that is based on Lagrangian backward tracing of the cells as described in section 4.2, similar to the MoF method (Dyadechko and Shashkov [37], Ahn and Shashkov [9]). However, the computation of the  $\alpha_c^{n+1}$

is not based on the intersection between the backward traced cell and the **PLIC** interface at the current time step  $\mathcal{H}_{\alpha,c}^n$ . Instead of using the area of the material polygon to compute the volume fraction value at the next time step  $\alpha_c^{n+1}$ , the material polygon is tracked forward in time and used for the  $\alpha_c^{n+1}$  computation. Explicit tracing of all points of the material polygon using an **RK** scheme leads to errors in mass conservation, as velocities evaluated at the material polygon points are not divergence free. Zhang and Liu [154], however, do not mention interpolation of the velocity field  $\mathbf{u}(x, t)$  at the points of the material volume in the text, they only mention the **RK** temporal displacement integration scheme as a possible source of errors in the volume conservation. It is not clear if the exact velocity  $\mathbf{u}(x, t)$  is evaluated for the points of material polygons, in which case the interpolation errors are completely avoided. After the forward tracing of material polygons, complex operations are required in order to correct non-convex material polygons to ensure volume conservation ([154, section 3.2]) and explicitly handle topological changes using merging algorithms for polygons ([154, section 3.3]). Verification results that involve temporally and spatially varying velocity field show that the **PAM** method is more accurate than **EMFPA** on structured equidistant meshes, however it is volume conservative within  $Ev \in [1e-08, 1e-05]$  depending on mesh resolution (e.g. [154, table 4]). Authors state that the method is approximatively 20% slower than **EMFPA** without reporting computational times either per time step or for the total simulation time for the used computer architecture. Extension of the calculations in 3D is stated to be a simple consequence of using Computational Geometry Algorithms Library (**CGAL**) for building the algorithms of the **PAM** method. Although **CGAL** is a powerful 3D geometrical library, the proposed approach of using Nef polyhedra for boolean operations is not efficient enough (cf. Hachenberger et al. [51]) for handling many polyhedron / half-space intersections required by a 3D implementation of the geometrical **VoF** method.

Ahn and Shashkov [9] have extended the **MoF** method proposed by Dyadechko and Shashkov [37] with local dynamic Adaptive Mesh Refinement (**AMR**) and show an overall second-order convergent solution for a set of standard verification cases, with different levels of local dynamic **AMR**. Adaptive **MoF** is implemented in 2D and the same problems in overshoots, undershoots and wisps are dealt with both local and global conservative error redistribution. Ahn et al. [11] have coupled the **MoF** method with the single-field two-phase **NS** equation system.

Zhang [151] proposes a fourth-order accurate representation of the Donating Region (**DR**), using the Donating Region Approximated by Cubic Splines (**DRACS**) to increase the accuracy of the approximation the flux volume. Instead of relying on linearized flux polyhedra or flux triangulation, Zhang [151] proposes the use of convex sets bounded with non-linear curves (hypothetically, surfaces in 3D) for solving equation (59). The **DRACS** scheme uses (E)**LVIRA** for piecewise linear interface reconstruction, so the fourth-order convergence is only shown for cases where no significant interface deformation occurs (solid body rotation, 2D shear with  $T = 0.5s$ ). In order to obtain an overall fourth-order accuracy for larger deformations, a higher-order volume conservative interface reconstruction is required. Zhang [151] motivates the development of the higher-order **DRACS** method by the

need for an accurate curvature calculation required for two-phase flows. However presented verifications cases with weak interface deformation or none whatsoever leave open a question if discontinuities that arise from topological changes of the interface in 3D (e.g. interface breakup and coalescence Popinet [105, figures 20 and 24]) can be handled, such that higher-order accuracy as well as solution robustness are maintained.

Le Chenadec and Pitsch [71] propose an elegant alternative **LE** geometrical **VoF** method that observes *material* volumes instead of *control* volumes, described here in more detail because of its elegance, simplicity of parallel implementation and direct applicability to unstructured meshes with cells of arbitrary shapes. They named their method Hybrid Lagrangian–Eulerian Method for Multiphase flow (**HyLEM**). A volume is considered to be a material volume if it evolves along Lagrangian trajectories given with the fluid velocity  $\mathbf{u}(x, t)$ . In this case, the volume always encloses the same set of material points as it evolves and no mass is entering or leaving the volume. For a material volume that contains two immiscible incompressible phases with constant densities  $\rho_1, \rho_2$ , it follows that the total mass of each respective phase does not change with the motion of the material volume. This can be stated for phase 1 as

$$\lim_{\delta\tau \rightarrow 0} \frac{1}{dt} \left( \int_{V_m(\tau+\delta\tau)} \rho_1 I(\mathbf{x}(\tau + \delta\tau), \tau + \delta\tau) dx - \int_{V_m(\tau)} \rho_1 I(\mathbf{x}(t), t) dx \right) = 0, \quad (136)$$

Discretized, equation (136) takes on the following form

$$\mathcal{H}_{m,\alpha}^{n+1} \cap V_m^{n+1} = \mathcal{H}_{m,\alpha}^n \cap V_m^n. \quad (137)$$

Motion of incompressible fluid phases can be prescribed by a velocity field if it satisfies  $\nabla \cdot \mathbf{u}(\mathbf{x}, t) = 0$ . If this is the case, the material volume does not change in magnitude, based on the Space Conservation Law (**SCL**) (Demirdzic and Peric [29]):

$$\frac{dV_m(t)}{dt} = \int_{\partial V_m(t)} \mathbf{u}(\mathbf{x}, t) \cdot \mathbf{n} d\sigma = \int_{V_m(t)} \nabla \cdot \mathbf{u}(\mathbf{x}, t) dx = 0. \quad (138)$$

Since the volume enclosed by the cell  $\mathbf{C}_c$  can be observed as a material volume  $V_c$ ,

$$V_m^{n+1} = V_m^n + \int_{\tau}^{\tau+\delta\tau} \frac{dV_m(t)}{dt} dt, \quad (139)$$

from where a very important condition

$$V_m^{n+1} = V_m^n \quad (140)$$

is formulated, that must be satisfied on the discrete level. The equation (140) is used and equation (137) is then divided with say  $V_m^{n+1}$ , that then leads to

$$\alpha_m^{n+1} = \alpha_m^n = \alpha_c^n. \quad (141)$$

Equation (141) leads to the elegant idea of Le Chenadec and Pitsch [71] for computing  $\alpha_c^{n+1}$ . First,  $V_m^{n+1}$  is computed by sweeping forward  $V_c$  along discrete Lagrangian trajectories (*forward tracing*). Equation (141) then allows the interface to be simply reconstructed in every  $V_m^{n+1}$ , resulting immediately with a **PLIC** interface in the new time step:  $\mathcal{H}_{m,\alpha}^{n+1}$ . However, in order to continue forward with the computation for  $n+2$ ,  $\alpha_c^{n+1}$  needs to be determined. It is important to note that  $\alpha_c^{n+1} \neq \alpha_m^{n+1}$ . The computation of  $\alpha_c^{n+1}$  can be formulated as the intersection between the new **PLIC** interface and the control volume  $V_c$ :

$$\alpha_c^{n+1} = \frac{1}{V_c} \text{volume}(\mathcal{H}_{i,\alpha}^{n+1} \cap_{i \in N_{c,\mathcal{H}_{i,\alpha}^{n+1}}} V_c^n), \quad (142)$$

such that

$$N_{c,\mathcal{H}_{i,\alpha}^{n+1}} = \{m : \text{volume}(\mathcal{H}_{m,\alpha}^{n+1} \cap V_c^n) > 0\}. \quad (143)$$

This idea is very elegant because the material volumes make it possible to use solely the mesh motion and interface reconstruction in order to obtain the **PLIC** interface in the new  $n+1$  time step. Its parallel implementation is simple as it requires a single exchange of the new interface elements ( $\mathcal{H}_{m,\alpha}^{n+1}$ ) across process boundaries, whose topology does not change with mesh motion. However, there are at least three difficulties in maintaining volume conservation. Note that equation (142) ensures exact numerical boundedness. First, the condition given by equation (140) is not upheld without corrections of  $V_m^{n+1}$  for volume conservation if discrete interpolated velocities are used to compute  $V_m^{n+1}$  by displacing  $V_c^n$ . Second, mapping  $(m, n+1) \rightarrow (c, n)$  required by equation (142) is surjective, and the **PLIC** interface reconstructed on  $V_m^{n+1}$  has discontinuities that may cause volume conservation errors when computing intersections in equation (142). Third, a 3D implementation will generate  $V_m^{n+1}$  as polyhedra with non-planar faces, so the reconstruction algorithm needs to be extended for interface positioning within such polyhedra. Le Chenadec and Pitsch [71] have left the 3D method generalization as future work and have shown for 2D Cartesian meshes that a second-order mass conservation error can be obtained by using higher-order **RK** schemes for the integration of mesh point displacements. From numerical experiments, the authors have come to the conclusion that volume conservation errors cancel out when the solutions of forward and backward Lagrangian backtracing are averaged, resulting in a combined forward/backward integration scheme for  $\alpha_c^{n+1}$ . Le Chenadec and Pitsch [71] reported third-order accurate volume conservation errors for the trapezoidal rule for cases with strong interface deformation within  $[1e-14, 1e-05]$ .

Zhang and Fogelson [153] propose the improved Polygonal Area Mapping Method (**iPAM**), a fourth-order accurate 2D method, that replaces the approximation of discrete Lagrangian trajectories with cubic splines in the **DRACS** scheme with displacements that are split into line segments that result from the sub-time stepping used for the point displacement integration. Impressive results presented by the authors show that the **iPAM** has the highest absolute accuracy among all other 2D geometrical **VoF** methods as well as a stable fourth-order convergence. However, already the 2D **iPAM** method is significantly more complex than other

geometrical **VoF** methods. In order to obtain fourth-order accuracy, additional marker points are added to the  $2D$  interface, that are then tracked along discrete Lagrangian trajectories integrated using the fourth-order accurate **RK** method. Consequently, **iPAM** method requires complex explicit topological operations to be performed on the interface to handle merging and coalescence, similar to those used by the Front Tracking method (cf. Tryggvason et al. [138]). Complex algorithms are already required for  $2D$  calculations to ensure the fourth-order of accuracy, and their extension for  $3D$  calculations is not straightforward: the authors again propose that the  $3D$  extension results from the simple use of Nef polyhedra and related boolean operations implemented in **CGAL**. As already pointed out, the execution times are already inadmissible for the geometrical **VoF** method: 1000 points in the intersection result cost  $\approx 10$  seconds, as reported by Hachenberger et al. [51]. Instead of relying on conservative error redistribution, material polygons are manipulated directly to ensure mass conservation. Complex algorithms that adjust material polygons by manipulating edges and adding/removing points used for ensuring local mass conservation are used. The authors state that local volume conservation cannot be fulfilled for some cells (Zhang and Fogelson [153, footnote, page 2370]), however they disregard this issue because of the fourth-order convergence of the geometrical volume fraction error. The volume (mass) conservation error is not reported the verification test cases. As for the computational efficiency, Zhang and Fogelson [153] state that **iPAM** is vastly more efficient than any other geometrical **VoF** method in terms of computational costs required to achieve the same accuracy. For  $h = \frac{1}{128}$ , they have calculated **iPAM** to be as much as 1650 times more efficient than any state-of-the art **VoF** and 100 times more efficient than the **AMR-MoF** method by Ahn and Shashkov [9] for the  $2D$  shear test case and  $h = \frac{1}{128}$ . However, the efficiencies are computed under the assumption that the CPU time consumption per time step of every other second-order geometrical **VoF** method is nearly equal to that of the **PAM** method and no High Performance Computing (**HPC**) measurements that support the theory are provided. Still, the **iPAM** method is very accurate and it delivers a stable higher-order convergence, which makes it an attractive candidate for a possible extension to  $3D$ , especially if the statements regarding its efficiency are confirmed by **HPC** measurements.

Parallel to the first implementation of the **UFVFC** scheme proposed in the present work (cf. [78]), Owkes and Desjardins [98] and Jofre et al. [65] have proposed Eulerian flux-based geometrical **VoF** methods that as well use unique vertex discrete Lagrangian trajectories. Unlike the vast majority of methods reviewed up to this point, both methods support  $3D$  computation. The method proposed by Jofre et al. [65] is directly applicable to unstructured meshes, whereas Owkes and Desjardins [98] state that the extension for unstructured meshes is straightforward. Both methods share the same correction of the flux volume  $V_f$  for volume conservation, covered extensively in section 6.3.3. Both methods rely on the **LVIRA** algorithm for the second-order accurate **PLIC** interface reconstruction and are parallelized using the domain decomposition model. They both show second-order convergent geometrical advection errors, are numerically bounded and volume conservative. The geometrical **VoF** method proposed in the present work was developed parallel to both methods, and its extensions give benefit to other flux-based geometrical



**VoF** methods as well, especially in terms of absolute accuracy (better flux volume triangulation, alternative reconstruction algorithms to **LVIRA**) and computational efficiency (reduction of intersection operations).

Comminal et al. [26] propose a 2D Cell-wise Conservative Unsplit (**CCU**) **LE** scheme. Contrary to Zhang and Liu [154], Zhang [151], Zhang and Fogelson [153], Zhang [152] that rely on the exact velocity to be prescribed at cell corner points as well as material polygon points, Comminal et al. [26] address the problem of interpolation. They show that the overall advection accuracy is influenced mostly by the integration of the discrete Lagrangian trajectories, and higher-order accurate velocity interpolation. For the Lagrangian backtracing of control volumes, Comminal et al. [26] rely on an explicit fourth-order accurate **RK** scheme, which is very interesting suggestion because the explicit nature of this high order scheme reinforces the coupling of the volume fraction equation with the rest of the two-phase **NS** equation system. Compared to **PAM** and **iPAM** schemes, **CCU** is much simpler because it replaces complex explicit manipulation of material polygons with a simple correction of the control volume pre-image for volume conservation. The same correction was already used by Jofre et al. [65], Owkes and Desjardins [98] for correcting flux polyhedra, and it is used in the method proposed in the present work as well. Additionally, topological changes do not have to be handled by the explicit manipulation of the geometrical interface, as is the case for the **PAM** method. Of course, the absolute accuracy and convergence of the **iPAM** and **PAM** is much higher, however, extending the **CCU** scheme to 3D unstructured meshes would involve significantly simpler geometrical operations. Comminal et al. [26] outline a possible extension for 3D calculations, that is also commented in the description of the geometrical **VoF** method proposed in the present work (cf. chapter 6).

An elegant alternative Eulerian flux-based geometrical **VoF** method implemented in 3D with support for arbitrary unstructured meshes was proposed recently by Roenby et al. [113], the *isoAdvector* scheme. The *isoAdvector* scheme avoids 3D intersection operations resulting from Lagrangian tracing by **LE** methods or from fluxed phase volume contribution calculations by Eulerian flux-based methods. The dimension of intersection operations is reduced from 3D to 2D because the *isoAdvector* scheme approximates the solution of the volume fraction equation as

$$\alpha_c^n + 1 = \alpha_c^n - \frac{0.5}{V_c} \left( \sum_f F_f^n \alpha_f^n + \sum_f F_f^{n+1} \alpha_f^{n+1} \right) \delta t \quad (144)$$

where  $\alpha_f$  is the *face fraction* given by the intersection between a linear approximation of the fluid interface on a face  $\mathbb{F}_f$ . The linear interface approximation is similar to the one proposed by Hernández et al. [54] for their Conservative Level Contour Interface Reconstruction (**CLCIR**). In order to achieve second-order accuracy,  $\alpha_f(t)$  is polynomially extrapolated from values given by the intersection between the linear fluid interface and the face  $\mathbb{F}_f$ . Positions of the interface (*iso-face*) are computed by tracing the linear interface element forward in time in a Lagrangian way, using the velocity at the cell center. The method is elegant and computationally inexpensive compared to other geometrical **VoF** methods, because the computation

of numerical fluxes by  $\alpha_f(t)$  completely avoids complex geometrical operations outlined in this section for other methods. As a consequence of using  $2D$  operations for flux calculations, the computational efficiency of the isoAdvect vector scheme is much better than for all other aforementioned geometrical **VoF** methods. Algorithm parallelisation using the domain decomposition approach is the same as for any Eulerian flux-based geometrical **VoF** method. As for the absolute accuracy, Roenby et al. [113] have compared the results for the  $3D$  deformation case against the results of Liovic et al. [73] and Hernández et al. [54]. Reported isoAdvect errors (Roenby et al. [113, page 21]) [ $3e-03$ ,  $6.4e-04$ ,  $1.2e-04$ ] for the respective meshes with  $[64^3, 128^3, 256^3]$  volumes are worse than those generated by the **PCFSC** from Liovic et al. [73, table 7] on finer meshes. Roenby et al. [113] have proposed the computation of phase fluxes using simpler geometrical operations than those that are required for other geometrical **VoF** methods, which represents an avenue worth investigating further.

### 5.3 CONCLUSIONS

The numerous methods used for the reconstruction of the **PLIC** interface and the advection of the volume fraction field outlined in sections 5.1 and 5.2 show how active geometrical **VoF** methods are being researched. The majority of higher-order methods are still being developed in  $2D$  and for structured Cartesian methods. Main obstacles in developing accurate, robust and efficient (both serial and parallel) geometrical **VoF** methods in  $3D$  with support for unstructured meshes are: geometrical operations, computational complexity behind the unstructured mesh topology, modular and yet highly computationally efficient software design. A set of three-dimensional geometrical operations specialized for **VoF** methods that are robust, accurate and yet computationally very efficient are required. Unstructured mesh topology complicates the direct access to mesh elements on available on structured meshes. A professional software development practice that ensures a modular and yet efficient implementation of both geometrical and transport algorithms is necessary. Accurate, robust and conservative geometrical **VoF** methods rely on consistent combinations of diverse sub-algorithms on unstructured meshes such as: displacement integration, flux / cell volume correction, intersection, convex hull, stencil computation, etc. It must be possible to exchange those sub-algorithms freely without a large computational and programming overhead and in order to deal with the methodological uncertainty. Although often claimed to be straightforward or even trivial, an extension of a geometrical **VoF** method to support  $3D$  computation on unstructured meshes is rather complex, which is easily confirmed by the number of publications that actually do support  $3D$  operations.



---

DEVELOPED METHOD

---

This chapter covers the details of the proposed **LE** flux-based geometrical **VoF** method.

Geometrical Volume-of-Fluid methods are currently being actively developed by the scientific community. Between the **LE** and the **LE** flux-based method, the decision was made to improve the **LE** flux-based method because of its inherent volume conservation property, simpler parallel algorithm implementation, and the fact that a flux-based method fits well within the unstructured **FV** method used by the OpenFOAM software platform for Computational Fluid Dynamics (**CFD**) (Weller et al. [147], Jasak and Jemcov [62]), which was chosen for the method implementation.

The novelty of the proposed method lies foremost in the use of unique mesh-point (cell corner) velocities for approximating backward Lagrangian trajectories. The motivation behind the usage of point velocities lies in the resulting absence of an overlap between adjacent flux polyhedra. This ensures volume conservation as, hypothetically, no amount of the phase flux volume can be fluxed twice over a face of a mesh cell.

Actually, volume conservation can only be considered as an inherent property of the **LE** flux-based method so far as the geometrical flux volume corresponds exactly to the volume given by the temporal integral of the volumetric flux. A correction resulting from this constraint was recently imposed by Comminal et al. [26] also for a **LE** geometrical **VoF** method (their 2D **CCU** scheme) to ensure volume conservation. In chapter 5, other **LE** methods are outlined that correct the swept cells to maintain volume conservation in one way or the other. In order to represent the flux polyhedron as accurately as possible, a new triangulation method for the flux polyhedron is proposed within the scope of this work. Visualization of flux polyhedra that fail to be corrected properly for volume conservation for the standard volume fraction advection cases has led to the conclusion that there is a strong relationship between the overall accuracy of the **LE** flux-based geometrical **VoF** method and the approximation of the flux polyhedron. This is confirmed by the results of the standard volume fraction advection verification tests described in chapter 11.

Additional to improving volume conservation, the discrete Lagrangian trajectories reconstructed with cell corner velocities represent more accurately the discrete flow map than the Lagrangian trajectories that result from geometrical simplifications of the flux polyhedra. When point velocities are used, the point displacements that build discrete Lagrangian trajectories are a direct result of a numerical approxi-

mation of the integrated point velocity over a time step  $\delta t$ . On the other hand, when the flux polyhedron is simplified in order to simplify geometrical operations, both the directions and the lengths of point displacements lose the connection with the point velocity integral. As a consequence, an accurate reconstruction of discrete Lagrangian trajectories using point velocities results in better absolute overall accuracy for the volume fraction transport algorithm. In this chapter, the methods that support the use of the cell corner velocities for the volume fraction transport are covered in detail.

Within this work, a single-step second-order accurate explicit temporal integration is proposed for both the discrete Lagrangian displacements and flux volumes  $V_f$ ; a different approach than often used **RK** integration schemes. This temporal integration replaces the first-order accurate equation (61) used in chapter 4 to outline the geometrical **VoF** method with an explicit single-step second-order accurate approximation.

An often disregarded aspect of a geometrical **VoF** method is the additional complexity required to extend the method for parallel algorithm execution, be it with a shared memory, or domain decomposition parallel model. For brevity, this aspect will be shortly named as: *method parallelization* or simply *parallelization* from this point on. Parallelization becomes very important as soon as the focus of the method application shifts from simple geometries and prescribed velocity fields used for method verification, to **DNS** of two-phase flows in geometrically complex solution domains of technical importance. In terms of method parallelization, the Euler flux-based geometrical **VoF** method is very straightforward, as it relies only on exchanging outgoing scalar phase flux volume  $V_f^\alpha$  fields across process boundaries. As the **LE** geometrical **VoF** method, **LE** flux-based geometrical **VoF** method, because of its explicit nature, lends itself to the *interleaving of computation with communication*, which leads to less time being spent by parallel tasks in waiting for the messages to be received. As for the shared memory parallelism, the reconstruction of the interface as well as the calculation of phase flux volumes  $V_f^\alpha$  does not incur any race conditions, which results in straightforward load balancing when shared memory parallelization is used.

Two interface reconstruction algorithms are used within this work that are second-order accurate, support unstructured meshes, do not introduce additional advected variables, and incur a smaller communication overhead.

The accuracy of the initialization algorithm for the volume fraction field  $\alpha_c$  determines the maximal absolute accuracy that can be achieved when solving the volume fraction equation (71). Three improvements are proposed for the existing initialization algorithm for the volume fraction field within this work. First, the initialization is based on the intersection operations between arbitrary shaped polyhedra, that increases the accuracy of initial volume fraction values. Second, serial reductions of intersection operations are introduced that save computational time. Third, the intersection operations are parallelized using the hybrid (message passing and shared memory) approach to parallelization. Additionally, a new initialization algorithm is proposed that computes a volume fraction field from the intersection between a surface and a volume mesh.

## 6.1 VOLUME FRACTION INITIALIZATION

The requirement of the volume fraction initialization algorithm for the initial interface reconstruction is covered in section 4.1.2. In this section, the developed volume fraction initialization algorithms are outlined.

Initializing volume fraction by representing the initial interface with a mesh (*interface mesh*)  $\Gamma_h$  was proposed for their MoF method by Ahn and Shashkov [7, cf. appendix D]. In the first step, a set of interface cells are defined in the solution domain  $\Omega_h$  as

$$N_c^i = \{c : \exists(\mathbf{p}, \mathbf{q}) \in \mathbf{C}_c, \mathbf{p} \text{ inside } \Gamma_h, \mathbf{q} \text{ not inside } \Gamma_h\}, \quad (145)$$

where  $\mathbf{p}$  and  $\mathbf{q}$  are points of the cell  $\mathbf{C}_c$ . Different algorithms have been developed to test if  $\mathbf{p}$  is inside  $\Gamma_h$  that are of practical use for large values of  $|\Gamma_h|, |\Omega_h|$ . To ensure short calculation times on 3D unstructured meshes with large initial interfaces, Ahn and Shashkov [5, 7] relied on the *ray crossing* version of a *point-in-polyhedron* algorithm described by O'Rourke [96, section 7.5] to compute  $\mathbf{p}$  inside  $\Gamma_h$ . To substantiate the decision for altering the initialization algorithm proposed by Ahn and Shashkov [5, 7], the ray crossing point-in-polyhedron algorithm of O'Rourke [96] is outlined in algorithm 1.

---

**Algorithm 1** Ray crossing point-in-polyhedron by O'Rourke [96]

---

```

1: while true do
2:   compute bounding radius  $R$ 
3:    $\mathbf{r}_0 =$  random ray of length  $R$ 
4:    $\mathbf{r} = \mathbf{q} + \mathbf{r}_0$ 
5:   crossings = 0
6:   for triangle  $\mathcal{T} \in \Gamma_h$  do INTERSECT( $\mathcal{T}, \mathbf{q}, \mathbf{r}$ )
7:     if degenerate intersection then
8:       re-enter while loop
9:     else
10:      increment crossings
11:    end if
12:  end for
13:  if crossings is odd then
14:    return  $\mathbf{q}$  is inside  $\Gamma_h$ 
15:  else
16:    return  $\mathbf{q}$  not inside  $\Gamma_h$ 
17:  end if
18: end while

```

---

O'Rourke [96] states that algorithm 1 is of complexity  $\mathcal{O}(cn)$ , where  $c$  is the number of times the **while** loop has been re-entered to disregard a special case, and  $n = |\Gamma_h|$ . Experiments show that  $\bar{c} \approx 1$  which makes algorithm 1 linear in terms of the size of the interface mesh  $n = |\Gamma_h|$ . If algorithm 1 is then used to categorize all points of the domain  $\Omega_h$  to identify interface cells  $N_c^i$  using equation (145), the overall complexity of evaluating  $N_c^i$  is of quadratic complexity  $\mathcal{O}(mn)$  where  $m = |\mathbb{P}|$  and  $n = |\Gamma_h|$ .

For interface cells  $N_c^i$  detected with equation (145) and algorithm 1, Ahn and Shashkov [7] compute the volume fraction using algorithm 2.

---

**Algorithm 2** Volume fraction initialization : Ahn and Shashkov [7]

---

```

1: for cell  $c \in N_c^i$  do
2:   AABBC = axis aligned bounding box ( $c$ )
3:   compute random points  $\mathbf{p}$ ,  $\mathbb{P}_{AABBC} = \{\mathbf{p} : \mathbf{p} \in AABBC(c)\}$ 
4:    $n_c^p = |\mathbb{P}_{AABBC}|$ 
5:    $n_{obj}^p = 0$ 
6:   for point  $\mathbf{p} \in \mathbb{P}_{AABBC}$  do
7:     if  $\mathbf{p}$  is inside  $\Gamma_h$  then
8:        $n_{obj}^p = n_{obj}^p + 1$ 
9:     end if
10:  end for
11:   $\alpha_c = \frac{n_{obj}^p}{n_c^p}$ 
12: end for

```

---

Two conclusions can be drawn by examining algorithm 2. First, the computation of the volume fraction  $\alpha_c$  is approximative, as it is defined by the ratio of the number of points inside  $\Gamma_h$  and the total number of points that fill *the axis-aligned bounding box of an interface cell*  $c \in N_c^i$ , compared to using geometrical intersections to compute  $\mathbf{C}_c \cap \Gamma_h$ . Therefore, the accuracy of algorithm 2 depends on the number of points randomly distributed within the bounding box of the interface cell  $n_c^p$ . Moreover, using the points that fill an axis aligned bounding box of a cell to compute its volume fraction is an inaccurate representation of the volume fraction on tetrahedral meshes, where the axis aligned bounding box may differ strongly from the geometry of the tetrahedron (cf. figure 23). This error is avoided when hexahedral cells are used to discretize the domain  $\Omega_h$  and is lower for polyhedral cells. Second, algorithm 2 has the complexity of  $\mathcal{O}(|N_c^i|n_c^p|\Gamma_h|)$  when it comes to categorizing each point  $\mathbf{p} \in \mathbb{P}_{AABBC}$  using the ray crossing algorithm.

The number  $n_c^p$  can be considered a large constant in this case, as surely more than 10 points are required to accurately represent the fill level of the cell bounding box. Consequently, the total complexity of the algorithm proposed by Ahn and Shashkov [7] to compute the volume fraction is  $Oh((|\Omega_h| + |N_c^i|n_c^p)|\Gamma_h|)$  - if the complexity is analyzed in terms of the intersection between a triangle and a ray. As stated by Ahn and Shashkov [7],  $n_c^p$  is a small number compared to  $|\Omega_h|$ , however, it is multiplied with  $N_c^i$  which can be large. Of course, this is the worst case complexity, since the ray-triangle intersection can be avoided by introducing **AABB** tests as proposed by O'Rourke [96]. The initialization algorithm 2 requires additional bounding box tests to include cells of  $\Omega_h$  into  $N_c^i$  that cannot be identified by algorithm 1 because their size may be too large to resolve small features of  $\Gamma_h$ . However, this difference is ignored in this description of new initialization algorithms as they do not require it.

The main reason for changing the volume fraction initialization procedure is to obtain an exact volume fraction field. In order to achieve this, the initialization

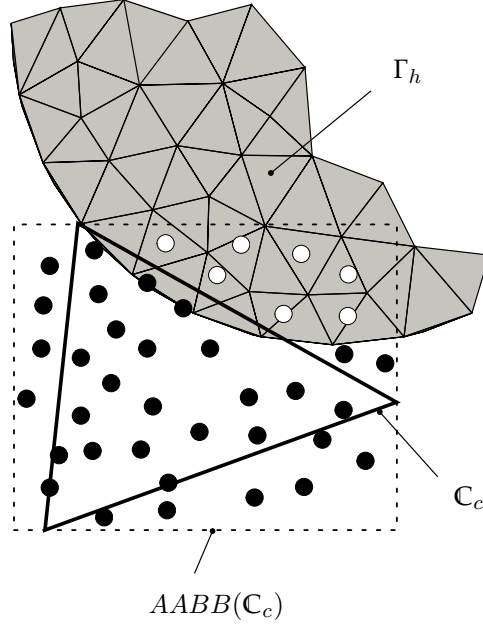


Figure 23: Schematic representation of the volume fraction initialization algorithm of Ahn and Shashkov [7]. The randomly generated points are schematically shown as circles of different colors: those that are inside  $\Gamma_h$  are of white color and they contribute to  $n_{obj}^p$  in algorithm 2. Obviously in this schematic case, all the points that are inside  $\Gamma_h$  are not inside the triangular cell  $\mathbf{C}_c$ ; however they are used to compute  $\alpha_c$ .

algorithms proposed in this thesis define the initial volume fraction using geometrical intersections

$$\alpha_c = \frac{|\mathbf{C}_c \cap \Gamma_h|}{|V_c|}, \quad (146)$$

where  $V_c$  is the volume of the cell  $\mathbf{C}_c$ . Two algorithms are proposed within this work that rely on equation (146) for exact calculation of the volume fraction: **CCI** and **SMCI** algorithms, outlined in algorithms 3 and 4. **CCI** algorithm 3 is a step in the right direction of computing an exact  $\alpha_c$  value. However, it has a complexity of  $\mathcal{O}(|\Omega_h|\bar{d})$  in terms of the 3D polyhedron intersection operation which costs much more than the triangle / ray intersection utilised by algorithm 2. The number  $\bar{d}$  is the number of cells  $\mathbf{C}_j \in \Gamma_h$  that intersect a cell  $\mathbf{C}_i \in \Omega_h$ . This number depends on the relative size of the cells in  $\Omega_h$  compared to the cells in  $\Gamma_h$ : it grows when the size of the cells in  $\Gamma_h$  is reduced. The first part of the algorithm reduces the complexity from  $\mathcal{O}(mn)$ , where  $m = |\Omega_h|$  and  $n = |\Gamma_h|$ , by pre-computing **AABBs** of cells in  $\Omega_h$  and  $\Gamma_h$ , which is a very cost-effective operation. In order to keep  $\bar{d}$  as small as possible,  $\Gamma_h$  mesh is created as an unstructured mesh, refined near its boundary that represents the fluid interface. This way, the complexity of the algorithm is reformulated as  $\mathcal{O}(k|N_b| + l|N_c^i|)$ , where  $N_b = \{\mathbf{C}_c : \mathbf{C}_c \in \Omega_h, \alpha_c = 1\}$ , with  $\bar{k} \approx 5$ ,  $\bar{l} \approx 100$  and of course  $|N_c^i| \ll |\Gamma_h| \ll |\Omega|$ . This reduction in effect makes algorithm 3 have linear complexity with respect to the cell/cell intersection

---

**Algorithm 3** CCI volume fraction initialization
 

---

```

1: Compute  $B_\Omega : \{B_\Omega^i : B_\Omega^i = \text{AABB}(\mathbf{C}_i), \mathbf{C}_i \in \Omega_h\}$ 
2: Compute  $B_\Gamma : \{B_\Gamma^j : B_\Gamma^j = \text{AABB}(\mathcal{T}_j), \mathcal{T}_j \in \Gamma_h\}$ 
3:  $B_{\Omega,\Gamma} = \{B_{\Omega,\Gamma}^i : B_{\Omega,\Gamma}^i = \emptyset\}$ 
4: for  $i \in [1, |B_\Omega|]$  do
5:   for  $j \in [1, |B_\Gamma|]$  do
6:     if  $B_\Omega^i \cap B_\Gamma^j \neq \emptyset$  then
7:        $B_{\Omega,\Gamma}^i = B_{\Omega,\Gamma}^i \cup j$ 
8:     end if
9:   end for
10: end for
11: for  $B_{\Omega,\Gamma}^i \in B_{\Omega,\Gamma}$  do
12:    $V_{\alpha,c} = 0$ 
13:   for  $j \in B_{\Omega,\Gamma}^i$  do
14:      $V_{\alpha,c} += \text{volume}(\mathbf{C}_i \cap \mathbf{C}_j), \mathbf{C}_i \in \Omega_h, \mathbf{C}_j \in \Gamma_h$ 
15:   end for
16:    $\alpha_c = \frac{V_{\alpha,c}}{V_c}$ 
17: end for

```

---

operation. Since the initial interface mesh  $\Gamma_h$  is not used for approximating a solution to a Partial Differential Equation (PDE), very strong refinement near the fluid interface boundary of  $\Gamma_h$  can be applied. With  $k = 100$  in interface cells, the initial interface is approximated with 100 cells of  $\Gamma_h$  within a single cell  $\mathbf{C}_i \in \Omega_h$ .

The initial interface mesh  $\Gamma_h$  that is refined near the interface ensures sufficient interface resolution for an accurate  $\alpha_c$  value. Very coarse cells away from the interface reduce the algorithm complexity, as the internal cells of  $\Omega_h$  are intersected with but a few cells of the interface mesh  $\Gamma_h$ . Figure 24 contains geometrical results from an actual algorithm execution on meshes from chapter 11. It shows that the volume fraction computation  $\alpha_c = \frac{\text{volume}(\Gamma_h \cap \mathbf{C}_c)}{V_c}$  in the case of the CCI algorithm involves many cell / cell intersections ( $\mathbf{C}_i \cap \mathbf{C}_j$  in algorithm 3) for interface cells (cell  $\mathbf{C}_c$  in figure 24), in order to ensure that the interface  $\Gamma_h$  is sufficiently resolved.

The CCI algorithm 3 computes exact volume fractions, and is amenable to multiple phases, by using multiple initial interface meshes  $\Gamma_h$ . Note that the linear complexity of algorithm 3 cannot be compared to the quadratic complexity of algorithm 2, for the obvious difference in the *unit operation*: the operation used to analyze the complexity of the algorithms. The unit operation is usually that operation where the majority of the computational time is spent by the algorithm. In asymptotic complexity analysis, algorithm operations are often assumed to take the same amount of time for a single algorithm, and complexity of different algorithms is compared based on this assumption as well. In practice, it is seldom the case that every operation of a single algorithm requires equal amount of work, and that two algorithms have the same unit operation. In this work, a unit operation is isolated using profiling tests with many problem sets of increased sizes. CCI algorithm was not implemented for its low complexity or execution times, the main goal was to ensure exact volume fraction initialization.

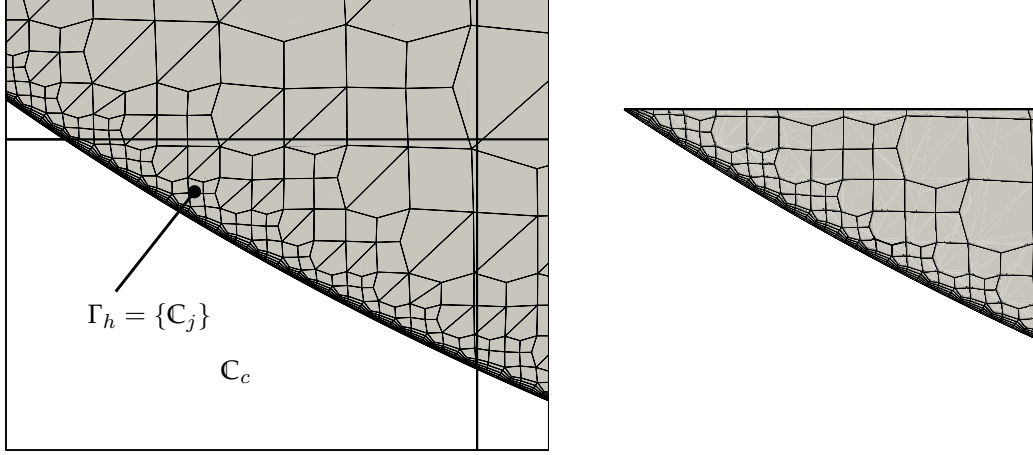


Figure 24: Volume fraction initialization of a verification case from chapter 11 using the CCI algorithm. The interface  $\Gamma_h$  is modeled as an unstructured volume mesh. The exact volume fraction value  $\alpha_c$  is computed using  $\mathbf{C}_c \cap \Gamma_h$ . left image shows an overlay of  $\Gamma_h$  and a cell  $\mathbf{C}_c$ , and the right image contains the actual result of a geometrical intersection  $\Gamma_h \cap \mathbf{C}_c$ .

Another significant contribution to the complexity of the CCI algorithm on a finer mesh comes from the intersection of all the bulk cells  $\alpha_c = 1$  with the initial interface  $\Gamma_h$ . Because those cells are filled completely, no intersections should be performed in order to set their volume fraction values  $\alpha_c = 1$ . Since the cells with  $\alpha_c = 1$  are filled with phase 1, the costly cell / cell intersection could be replaced by an algorithm that marks them as *inside cells* or *bulk cells*. For this purpose, the SMCI algorithm (outlined as algorithm 4) is proposed that performs inside-outside categorization using a spatial sub-division based on the octree data structure that supports fast search queries (Meagher [85], Mehta and Sahni [86]).

The SMCI algorithm models the initial interface not as a *volume mesh* as is the case for the CCI algorithm, but as a *surface mesh*. In this case, the interface mesh  $\Gamma_h$  is an unstructured mesh of mutually connected triangles, based on indirect addressing (chapter 3). In order to concentrate search operations to a near vicinity of the interface, the SMCI algorithm initializes a *squared search distance* for every cell  $\mathbf{C}_c \in \Omega_h$  as

$$\psi_c = \frac{1}{|\mathbf{C}_c|} \sum_f \|\mathbf{d}_f\|^2, \quad (147)$$

where  $\mathbf{d}_f$  is the difference between the cell centers:

$$\mathbf{d}_f = I_p(N_f) - I_p(P_f). \quad (148)$$

Furthermore, the vertices of  $\Gamma_h$  are inserted as leaf elements in an octree that subdivides the AAB of  $\Gamma_h$ . To each cell  $\mathbf{C}_c \in \Omega_h$ , a *cell sphere* is associated:

$$\mathcal{B}_c = (\mathbf{x}_c, r_c) : r_c = \sqrt{\psi_c}. \quad (149)$$



---

**Algorithm 4** SMCI volume fraction initialization

---

```

1:  $\alpha_c = 0$ 
2: Compute squared cell search distances  $\psi_c$  using equation (147).
3: for  $\mathbf{x}_c \in \Omega_h$  do
4:   Octree: find triangle  $\mathcal{T}_c \in \Gamma_h$  nearest to  $\mathbf{x}_c$  within  $\psi_c$ .
5:   if  $\mathcal{T}$  found then
6:     Set distance( $\mathbf{x}_c, \mathcal{T}_c$ ) =  $\phi_c$  using equation (150).
7:   end if
8: end for
9: Solve diffusion equation (151) for  $\phi_c$ .
10: for  $\mathbf{C}_c \in \Omega_h$  do
11:   if  $\exists \mathcal{T}_c$  then
12:     Octree: find triangles  $N_c^{\mathcal{B}_c}$  (equations (149) and (152))
13:     Compute the cell triangulation  $T_c(\mathbf{C}_c)$ 
14:     for  $\mathcal{T} \in N_c^{\mathcal{B}_c}$  do
15:        $T_c = T_c \cap \mathcal{T}$ 
16:     end for
17:      $\alpha_c = \frac{\text{volume}(T_c)}{V_c}$ 
18:   else
19:     if  $\phi_c < 0$  then
20:        $\alpha_c = 1$ 
21:     end if
22:   end if
23: end for

```

---

The cell sphere is used by the octree data structure to exclude those cells from the nearest distance query operations, whose spheres  $\mathcal{B}_c$  lie completely outside of the AABB of the octree. Those cells whose cell sphere intersects  $\Gamma_h$  are associated with a signed distance from a *nearest triangle*, with a signed distance to the nearest triangle defined as

$$\phi_c = (\mathbf{x}_c - \mathbf{v}_{\mathcal{T},c}) \cdot \mathbf{n}_{\mathcal{T},c}. \quad (150)$$

A result of this part of the SMCI algorithm 4 is a signed distance field in the narrow band of  $\Gamma_h$ , with the width of the narrow band determined by  $r_c$ . Such a distance field can be used to propagate the inside-outside information for every point in the domain, by solving a diffusion equation for the signed distance,

$$\partial_\tau \phi - \nabla \cdot (\lambda \nabla \phi) = 0, \quad (151)$$

where  $\lambda$  is an optional signed distance diffusion coefficient ( $\lambda = 1$  is used in this work) and  $\tau$  is the pseudo-time step (the time step used for advection can also be applied). The solution of the equation (151) is based on the unstructured FV method outlined in chapter 3. Since equation (151) is a diffusion equation, the physical end time of the solution determines the intensity of the diffusion for the signed distance field. In other words, equation (151) should be solved in such a way that the  $\phi$  field remains mostly undisturbed, with only the sign of the distance being propagated throughout the domain. The sign propagation is then used to



identify the inside and outside cells. In this initial implementation of the algorithm, this has been simply achieved by using a large value for the tolerance of the linear solver used to solve the linear algebraic system resulting from the unstructured **FV** method discretization of equation (151). The diffused signed distance field is then used by the **SMCI** algorithm 4 to set the values of cells fully occupied by phase 1 to  $\alpha_c = 1, \phi_c < 1$ , thus avoiding unnecessary cell intersections performed for the full cells by the **CCI** algorithm.

The next step of the **SMCI** algorithm is to perform the  $\Gamma_h \cap \Omega_h$  to determine the volume fraction  $\alpha_c$  for partially filled cells. In order to reduce the computational complexity of such an operation, the octree data structure is used to compute a set of triangles  $N_c^{\mathcal{B}_c} \subseteq \Gamma_h$  that intersect individual cell spheres,

$$N_c^{\mathcal{B}_c} = \{\mathcal{T} \in \Gamma_h : \mathcal{T} \cap \mathcal{B}_c \neq \emptyset\}. \quad (152)$$

Using equation (152), the **SMCI** geometrically computes the exact volume fraction given by  $\Omega_h \cap \Gamma_h$  for every partially filled cell. In order to increase the accuracy of the intersection  $\mathcal{C}_c \cap N_c^{\mathcal{B}_c}$ , each cell that has  $N_c^{\mathcal{B}_c} \neq \emptyset$  is decomposed into tetrahedra (*triangulated*). Geometrical and field results of the **SMCI** volume fraction initialization is shown for a verification case in figure 25.

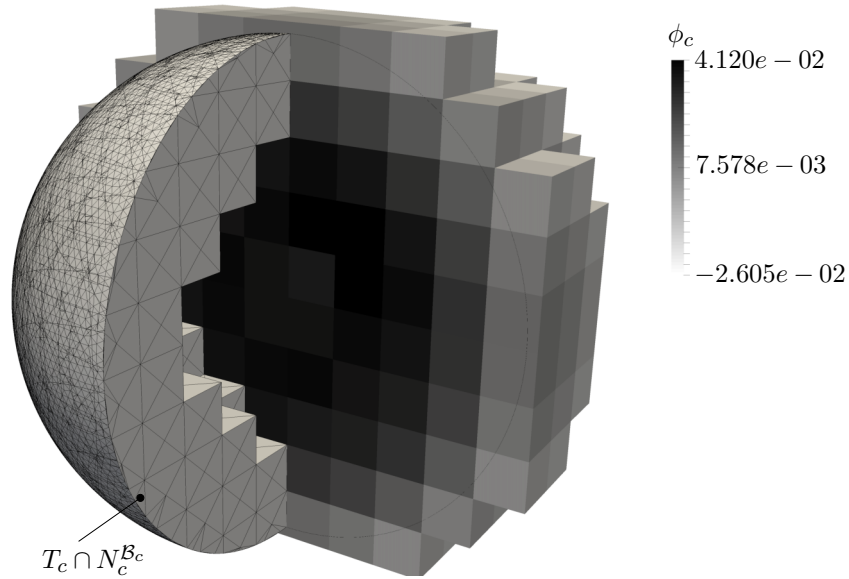


Figure 25: The **SMCI** algorithm intersects cell sphere triangles  $N_c^{\mathcal{B}_c}$  with the triangulation of a cell  $\mathcal{C}_c, T_c$  to set the volume fraction value  $\alpha_c$ . The signed distance  $\phi_c$  field obtained by solving equation (151) is used to mark the cells that are inside, or outside  $\Gamma_h$ . In this case, the narrow band of cells of  $\Omega_h$  that surround  $\Gamma_h$  is 3 cells wide.

The **SMCI** algorithm replaces the characterization of all points as internal or external with respect to  $\Gamma_h$  performed by the algorithm of Ahn and Shashkov [7, 5], as well as the unnecessary cell intersection operations of the **CCI** algorithm. Because the steps of **SMCI** differ strongly from each other, profiling was used to analyze the serial computational bottleneck of this algorithm. For all the verification cases, the intersection  $T_c \cap N_c^{\mathcal{B}_c}$  is the step where the algorithm spends  $> 90\%$  of the

computation time, while the signed distance calculation and the solution of the signed distance equation (151) both require a few % of the computational time on average.

Both **SMCI** and the **CCI** algorithm are parallelized using the hybrid shared memory and domain decomposition parallel programming model (hybrid OpenMP and Open MPI parallelization). Both algorithms can be made embarrassingly parallel on the level of their unit operations, because no race conditions are present and no communication is required across process boundaries. In order to achieve this, the complete  $\Gamma_h$  must be used by all threads and processes. As  $\Gamma_h$  is strongly refined near the interface with the **CCI** algorithm and since  $\Gamma_h$  is modeled as a surface mesh by the **SMCI** algorithm, its small size makes such data distribution possible without introducing unnecessarily high memory use. This is of course valid only for the verification cases presented in this thesis. General use that involves intersection of arbitrarily complex surface meshes  $\Gamma_h$  with arbitrarily complex solution domains  $\Omega_h$  is left as a subject of future work.

At this point an exact volume fraction field  $\alpha_c$  is considered to be computed by either the **CCI** or the **SMCI** algorithm, that is then used to first reconstruct the **PLIC** interface, and then advect the volume fraction field. Those next steps of the proposed geometrical **VoF** method are covered in the following sections.

## 6.2 INTERFACE RECONSTRUCTION

As outlined in section 4.1.2, the interface reconstruction algorithm is responsible for reconstructing a piecewise-planar interface approximation using the so-called **PLIC** algorithm. The **PLIC** algorithms proposed in this thesis reconstruct the interface on unstructured meshes that contain cells of arbitrary shape. The interface reconstruction algorithm consists of two main sub-algorithms: *interface orientation* and *interface positioning*.

The proposed gradient scheme as well as the **PLIC** reconstruction algorithm enhancements outlined in this section lead to improvements of the order of convergence for the interface verification tests covered in chapter 11.

### 6.2.1 Interface orientation and positioning

For the interface positioning, the algorithm of Ahn and Shashkov [8] is used as it offers a good balance of accuracy, computational efficiency and ease of implementation. Modular software design of the proposed geometrical **VoF** method described in section 6.6 makes it possible to easily exchange the iterative interface positioning policy of Ahn and Shashkov [8] with an alternative semi-iterative analytic positioning policy, like the one proposed by Diot et al. [35], Diot and François [34].

As a result of the consideration of the works done by Mavriplis [83], Aulisa et al. [14], Ahn and Shashkov [5], Correa et al. [27], two gradient schemes were implemented to estimate the interface normal  $\mathbf{n}_c$ : the Least Squares Gradient (**LSG**) and the Inversed Distance Weighted Gauss Gradient (**IDWGG**). The **LSG** scheme is

a known method of computing gradients on unstructured meshes and is described in detail by Mavriplis [83] and Correa et al. [27], so its description is omitted here and placed in appendix A.

The proposed **IDWGG** is a modification of the gradient scheme proposed by Aulisa et al. [14]. Values of the volume fraction  $\alpha_c$  at cell corner points are computed as

$$\alpha_p = \sum_{pc \in N_{pc}} w_{pc} \alpha_{pc} \quad (153)$$

such that *point cells*  $N_{pc}$  represents the set of all cells that are joined at point  $\mathbf{p}_p$ ,

$$N_{pc} = \{c : \mathbf{p}_p \cap \mathbf{C}_c = \mathbf{p}_p\}, \quad (154)$$

and the **IDW** point-cell inverse distance weight is given as

$$w_{pc} = \frac{1}{\|\mathbf{x}_{pc} - \mathbf{x}_p\|^p} \sum_{pc \in N_{pc}} \frac{1}{\|\mathbf{x}_{pc} - \mathbf{x}_p\|^p}. \quad (155)$$

$p = 1$  is used in the present work. Once the mesh point volume fraction value  $\alpha_p$  is determined, a face-centered value is computed as

$$\alpha_f = \frac{1}{N_{fp}} \sum_{fp \in N_{fp}} \alpha_{fp}. \quad (156)$$

The face-centered volume fraction value is then used to compute the gradient as

$$\nabla \alpha_c \approx \nabla_c^{IDW} \alpha_c = \sum_f \alpha_f \mathbf{S}_f. \quad (157)$$

The discrete gradient of the volume fraction field then results in the estimate of the interface normal,

$$\mathbf{n}_c = \frac{\nabla_c^{IDW} \alpha_c}{\|\nabla_c^{IDW} \alpha_c\|}. \quad (158)$$

Since  $\alpha_p$  contains contributions from surrounding point-adjacent cells (equation (153)), the final result given by equation (158) contains the error of the **IDW** interpolation. The **IDW** interpolation smooths out the sharp  $\alpha_c$  field, which in effect stabilizes equation (158) on very fine resolutions that would otherwise resolve the sharp jump in  $\alpha_c$  and lead to issues in numerical stability. Of course, the increase in stability of the **IDWGG** resulting from the **IDW** interpolation is countered by the loss of accuracy of the inverse distance weighting average. Another important reason exists for using the point-cell connectivity this way: the accurate finite difference formulation used by Aulisa et al. [14] for the mesh-point volume fraction values  $\alpha_p$  cannot be directly computed on unstructured meshes. That is the reason for relying on the point-cell connectivity given by the indirect addressing of the unstructured mesh outlined in chapter 3. Replacing the **IDW** averaging step is possible, however, it is not effective: better accuracy of the discrete gradient applied on a sharp field results in higher numerical instability on fine meshes.

The proposed **IDW** Gauss gradient scheme represents a contribution to the estimate of the interface orientation part of the Youngs' **PLIC** reconstruction algorithm on unstructured meshes. It accurately estimates the interface orientation on coarse unstructured meshes and maintains numerical stability when fine mesh resolution is used. This is supported by results of the verification tests.

At this point it can be assumed that the interface initial orientation is given either by **LSG** or **IDWGG** and the interface is positioned using the algorithm of Ahn and Shashkov [8]. In order to maintain second-order accuracy, an extension of the Youngs' **PLIC** reconstruction algorithm is required. Many such extensions that have been developed so far and could be extended to unstructured meshes are covered in detail in section 5.1. Next section covers modifications of existing second-order **PLIC** reconstruction algorithms that were found to be useful during numerical verification procedure, as well as new extensions of the **PLIC** reconstruction algorithm. Proposed extensions result in overall second-order error convergence and reduce the parallel communication overhead of their parallel implementation, important nowadays for solving large problems of technical complexity.

The Swartz algorithm resulting from the modification of the Swartz-Mosso algorithm by Dyadechko and Shashkov [37] is described in detail in section 5.1. The implementation and control parameters of the Swartz algorithm used to generate the results in chapter 11 differ somewhat from the algorithm described by Dyadechko and Shashkov [37], so algorithm 5 outlines the implementation used in this work.

Algorithm 5 is named in this text as Youngs' / Swartz Reconstruction algorithm (**YSR**) to distinguish it from the original Mosso-Swartz algorithm and its modification in the form of Swartz algorithm proposed by Dyadechko and Shashkov [37]. The **YSR** algorithm contains steps that are additional to those proposed by Dyadechko and Shashkov [37]. Reformulating the original Mosso-Swartz algorithm is based on excluding outer iterations and using only 4 inner normal correction loops, applied on every interface cell only when the interface normal  $\mathbf{n}_c$  forms an angle less than  $45^\circ$  with its neighbor. To avoid artificial smoothing of the interface for under-resolved curvature, and to exploit the second-order convergence of the Swartz algorithm where the interface is finely resolved, algorithm 5 also relies on the Youngs' algorithm in regions where the curvature is under-resolved.

Algorithm 5 starts by reconstructing the interface using the Youngs' algorithm with the **IDWGG** gradient scheme used for the normal orientation instead of the **LSG** scheme. The initial interface reconstruction results in the first set of interface polygons, their centroids and normal area vectors. Next, an average interface normal vector is computed for each interface cell with index  $c$  as

$$\bar{\mathbf{n}}_c = \frac{1}{\sum_f \|\mathbf{S}_f\|} \sum_f \mathbf{n}_f \|\mathbf{S}_f\|, \quad (159)$$

where linear interpolation is used for  $\mathbf{n}_f$  in this text although other ways of computing  $\mathbf{n}_f$  are possible. The average interface normal is used throughout the algorithm to check if the corresponding interface polygon is to be used to compute the contribution to the corrected normal  $\mathbf{n}_c^m$ .

---

**Algorithm 5** Youngs' / Swartz reconstruction
 

---

```

1: for  $n < n_{recs}$  do
2:   for  $m < n_{corr}$  do
3:      $\bar{\mathbf{n}}_c = \text{average}(\mathbf{n}_c)$ 
4:      $\bar{\mathbf{n}}_c = \frac{\bar{\mathbf{n}}_c}{\|\bar{\mathbf{n}}_c\|}, \mathbf{n}_c = \frac{\mathbf{n}_c}{\|\mathbf{n}_c\|}$ 
5:      $\mathbf{n}_c^m = \mathbf{n}_c$ 
6:     for  $c \in \{1..|\Omega_h|\}$  do
7:       if  $\alpha_c > \alpha_{c,corr}$  and  $\alpha_c < (1 - \alpha_{c,corr})$  then
8:         if  $(\mathbf{n}_c \cdot \bar{\mathbf{n}}_c) > \cos(\beta_{c,crit})$  then
9:           Compute the interface polygon  $\mathcal{Q}_c$ .
10:           $\mathbf{x}_{\mathcal{Q}_c} = \text{centroid}(\mathcal{Q}_c)$ 
11:          Compute the polygon centroid  $\mathbf{x}_{\mathcal{Q}_c}$ .
12:          Find all adjacent cells with an interface  $N_{c,\alpha_c}$ .
13:           $p = 1$ 
14:          for  $i \in N_{c,\alpha_c}$  do
15:            if  $\alpha_i > \alpha_{c,corr}$  and  $\alpha_i < (1 - \alpha_{c,corr})$  then
16:              if  $(\mathbf{n}_i \cdot \bar{\mathbf{n}}_c) > \cos(\beta_{c,crit})$  then
17:                Compute neighbor cell interface polygon  $\mathcal{Q}_{c,i}$ .
18:                 $\mathbf{x}_i = \text{centroid}(\mathcal{Q}_{c,i})$ .
19:                 $\mathbf{r} = (\mathbf{x}_i - \mathbf{x}_{\mathcal{Q}_c}) \times \mathbf{n}_c$ 
20:                 $q = \text{quaternion}(\mathbf{r}, 0.5\pi)$ 
21:                 $\mathbf{n}_c^m = \mathbf{n}_c^m + q\mathbf{r}\bar{q}$ 
22:                 $p = p + 1;$ 
23:              end if
24:            end if
25:          end for
26:           $\mathbf{n}_c^m = \mathbf{n}_c^m / p$ 
27:        end if
28:      end if
29:    end for
30:     $\mathbf{n}_c = \mathbf{n}_c^m, \forall c \in \{1..|\Omega_h|\}$ 
31:  end for
32:  Re-position all interfaces using new  $\mathbf{n}_c$ .
33: end for

```

---

The outer loop of the algorithm is controlled by the parameter  $n_{recs}$ , which is set to 1 for all the verification test cases reported in this work - same as for the Swartz algorithm by Dyadechko and Shashkov [37]. Since the normal reconstruction based on any gradient scheme introduces errors for cells that are almost full or almost empty, using the normals of such cells to improve the overall order of convergence is not possible: their instabilities amplify the error. Therefore, the **YSR** algorithm uses an additional filter,  $\alpha_{c,corr}$  in order to exclude almost full or almost empty cells from contributing to the corrected normal. For all results presented in chapter 11,  $\alpha_{c,corr} = 1e - 03$  was used.

Instead of checking face-adjacent interface normal vectors, the angle between the normal vector of the reconstructed interface  $\mathbf{n}_c$  and the average normal vector  $\bar{\mathbf{n}}_c$  is used to determine if for a cell  $c$ , the normal correction should be executed. The critical angle was set to  $\beta_{c,crit} = 20^\circ$ . This is a result of numerical experiments for the verification tests that involve strong interface deformation. An angle value of  $45^\circ$  applies normal modification also for those parts of the interface with the under-resolved curvature, in which case the second-order convergence provided by the Youngs' algorithm in under-resolved regions is lost.

The neighboring interface polygons are accessed from interface cells that are defined as

$$N_{c,\alpha_c} = \{d : \mathbf{C}_c \cap \mathbf{C}_d \neq \emptyset, \epsilon_r < \alpha_d < 1 - \epsilon_r\}, \quad (160)$$

resulting in a set of indexes to neighboring cells of the cell  $\mathbf{C}_c$  that share at least a single point and contain the **PLIC** interface. Other parts of the **YSR** algorithm are the same as the steps of the Swartz algorithm by Dyadechko and Shashkov [37]. However, one detail warrants further explanation: the number of internal iterations  $n_{recs}$  used when the **YSR** algorithm is coupled together with the volume fraction advection. Dyadechko and Shashkov [37] state that 4 corrections are sufficient to maintain the second-order convergence for the reconstruction. However, verification tests covered in chapter 11 required at least 10 internal normal corrections when the volume fraction advection is computed and the interface is significantly deformed by the prescribed velocity field.

In addition to the slightly modified Swartz algorithm (**YSR**), a new reconstruction algorithm is proposed in this thesis that increases the accuracy of the Youngs' reconstruction algorithm on fine meshes without introducing complex operations required for the parallel algorithm implementation: the **DG NR** reconstruction algorithm. The **DG NR** algorithm reduces the **PLIC** reconstruction error resulting from the gradient evaluation on finer meshes by reducing the characteristic gap between the adjacent **PLIC** interface polygons and its steps are outlined in algorithm 6. In its essence, the **DG NR** algorithm reconstructs the signed distance between the cell centers and the interface half-spaces as well as the signed distance between the mesh points (cell corner points) and the nearest interface half-spaces. The distances associated to the cell center and corner points are then used to compute the corrected interface normal vector, by computing the distance gradient within each cell using least-squares minimization. Since the signed distance between the interface half-space and the cell center is unique to a cell, only the cell corner point

distances that are associated with the process domain need to be exchanged in order to parallelize the **DG NR** algorithm using the domain decomposition model.

---

**Algorithm 6** Distance Gradient Normal Reconstruction (**DG NR**)
 

---

```

1: for  $n < n_{recs}$  do
2:   for  $m < n_{corr}$  do
3:      $\bar{\mathbf{n}}_c = \text{average}(\mathbf{n}_c)$ 
4:      $\bar{\mathbf{n}}_c = \frac{\bar{\mathbf{n}}_c}{\|\bar{\mathbf{n}}_c\|}, \mathbf{n}_c = \frac{\mathbf{n}_c}{\|\mathbf{n}_c\|}$ 
5:      $\mathbf{n}_c^m = \mathbf{n}_c$ 
6:     for  $c \in \{1..|\Omega_h|\}$  do ▷ Pre-compute least squares matrix inverse.
7:       if  $\alpha_c > \epsilon_r$  and  $\alpha_c < (1 - \epsilon_r)$  and  $W_{cp} = \emptyset$ . then
8:         for  $pc \in N_{pc}$  do
9:           Compute cell-point weights  $W_{cp}$ .
10:          Compute the inverse of the LS coefficient matrix  $\mathbf{L}_{pc}^{-1}$ .
11:        end for
12:      end if
13:    end for
14:    for  $c \in \{1..|\Omega_h|\}$  do
15:      if  $\alpha_c > \epsilon_r$  and  $\alpha_c < (1 - \epsilon_r)$  then
16:        Calculate cell centered signed distance  $\phi_c = \text{distance}(\mathcal{H}_{\alpha,c}, \mathbf{x}_c)$ 
17:      end if
18:    end for
19:    for  $p \in \{1..|\mathbb{P}|\}$  do ▷ Find minimal point-interface distance.
20:      Initialize minimal point signed distance  $\phi_{p,min} = 1e15$ 
21:      for  $pc \in N_{pc}$  do
22:        if  $\alpha_{pc} > \epsilon_r$  and  $\alpha_{pc} < (1 - \epsilon_r)$  then
23:          Calc. signed distance at the point  $p$ ,  $\phi_p = \text{distance}(\mathcal{H}_{\alpha,cp}, \mathbf{x}_p)$ 
24:          if  $|\phi_p| < \phi_{p,min}$  then
25:             $\phi_{p,min} = \phi_p$ 
26:          end if
27:        end if
28:      end for
29:    end for
30:    for  $cin\{1..|\Omega_h|\}$  do
31:      if  $\alpha_c > \epsilon_r$  and  $\alpha_c < (1 - \epsilon_r)$  then
32:        Compute the least squares distance source  $\mathbf{b}_c$ .
33:         $\mathbf{n}_c = \mathbf{L}_c^{-1} \mathbf{b}_c$ 
34:      end if
35:    end for
36:  end for
37:  Re-position all interfaces using new  $\mathbf{n}_c$ .
38: end for

```

---

### 6.3 VOLUME FRACTION ADVECTION

The **UFVFC** scheme covered in this section relies on using velocities evaluated at cell corner points (mesh points) to build the flux polyhedron. Using mesh point velocities results in a geometrically complex three-dimensional flux polyhedron: in

the general case, the flux polyhedron will have self intersecting, non convex and non planar faces.

The **UFVFC** scheme in its initial form [78] was not the only contribution to the vertex-based dimensionally un-split **LE** flux-based geometrical **VoF** method in 3D: parallel development was done by Jofre et al. [65] on unstructured meshes and Owkes and Desjardins [99] on structured Cartesian meshes. Other recent important contributions to the **LE** category of the geometrical **VoF** methods are discussed in chapter 5. Computational steps of the **UFVFC** scheme are outlined in algorithm 7.

---

**Algorithm 7** Unsplit Face-Vertex Flux Calculation (**UFVFC**)

---

```

1: Displace mesh points along discrete trajectories (section 6.3.1).
2: for  $f \in \{1..M_f\}$  do
3:   Compute the flux stencil  $N_{F_f}$  (section 6.3.2).
4:   if the face  $F_f$  is in narrow band then (section 6.3.2).
5:     if  $F_f < 0$  then
6:       Change the orientation of the face  $F_f$ .
7:     end if
8:     Build the flux polyhedron with volume  $V_f$  (section 6.3.3).
9:     Compute the geometrical phase flux volume  $V_f^\alpha$  (section 6.3.4).
10:  else
11:    Use the upwind phase flux volume.
12:  end if
13: end for

```

---

The sub-algorithms of algorithm 7 contain operations that are different from one mesh face  $F_f$  to another. The overall execution speed, accuracy, stability and robustness of the **UFVFC** scheme depends strongly on how those properties are supported by its sub-algorithms, as well as it depends on a valid sub-algorithm combination. Relevant details of individual sub-algorithms of the **UFVFC** scheme are covered in the following sections.

### 6.3.1 Point displacement integration

In order to integrate point displacements along discrete Lagrangian trajectories backward in time (*Lagrangian backward tracing*), velocities need to be computed at mesh points. In their very recent work, Comminal et al. [26] study the difference in the overall solution accuracy and conclude that the interpolation of the vertex velocities plays a significant role. That much so, that they employ a tricubic interpolation method in order to increase the overall accuracy of the Lagrangian backward tracing step. Owkes and Desjardins [99] mention only the possibility of interpolating vertex velocities one time and they do not provide information on the used interpolation scheme. Zhang and Fogelson [153] do not mention the problem of vertex velocity interpolation in their fourth-order accurate two-dimensional improved Polygonal Area Mapping Method (**iPAM**) scheme: they rely on an exact velocity field  $\mathbf{u}(\mathbf{x}, t)$  in order to numerically solve the volume fraction advection equation. Since the **UFVFC** scheme is to be coupled with a *second-order accurate* unstructured **FV** method and applied to problems modeled with second-order **PDEs**,



spatial interpolation used for the Lagrangian backward tracing was kept as simple as possible and yet capable of maintaining overall second-order convergence of all the verification tests. Therefore, one interpolation method was taken over from López et al. [75]: the **IDW** interpolation is used to obtain the point velocities

$$\mathbf{u}_p = \sum_{pc \in N_{pc}} w_{pc} \mathbf{u}_{pc}, \quad (161)$$

with the *linear IDW* cell point distance weight

$$w_{pc} = \frac{1}{\|\mathbf{x}_{pc} - \mathbf{x}_p\|} \sum_{\tilde{pc} \in N_{pc}} \frac{1}{\|\mathbf{x}_{\tilde{pc}} - \mathbf{x}_p\|}. \quad (162)$$

Additionally, an alternative, simple and easily parallelized, second-order accurate cell-point interpolation is proposed, that can be computed using an averaged Taylor series expansion of the velocity field  $\mathbf{u}_c$  from the cell center to every cell corner point according to

$$\mathbf{u}_p = \frac{1}{|N_{pc}|} \sum_{pc \in N_{pc}} \mathbf{u}_{pc} + \nabla_c \mathbf{u}_{pc} \cdot (\mathbf{x}_p - \mathbf{x}_{pc}) + \frac{1}{|N_{pc}|} \sum_{pc \in N_{pc}} \epsilon_{pc,T} + \epsilon_{pc,\nabla} \quad (163)$$

Second-order of accuracy of equation (163) depends on the order accuracy of the discrete gradient  $\nabla_c$ , since the truncation error of the linear Taylor series is  $\epsilon_{pc,T} = \mathcal{O}(\|\mathbf{x}_p - \mathbf{x}_{pc}\|^2)$ . The unstructured **FV** described in chapter 3 is formally second-order accurate, resulting in  $\epsilon_{pc,\nabla} = \mathcal{O}(\|\mathbf{d}\|^2)$ . Since  $\|\mathbf{d}\| \approx \|\mathbf{x}_p - \mathbf{x}_{pc}\|$ ,  $\forall pc$  and the average of the error over  $N_{pc}$  does not decrease its order of accuracy, equation (163) represents a second-order accurate cell-point interpolation on unstructured meshes. Equation (163) is a very simple way of accurately interpolating from cell centers to mesh points (cell corner points) on unstructured meshes because it does not involve assembling complex stencils on unstructured meshes that become computationally expensive and complex to both construct and exchange across process boundaries in a parallel implementation, especially if the mesh is dynamically topologically changed by the local dynamic Adaptive Mesh Refinement (**AMR**). A parallel implementation of equation (163) requires only that the point values at process boundaries are exchanged and averaged.

Two integration schemes are used for the mesh point displacements along discrete Lagrangian trajectories: the trapezoidal and the Taylor integration scheme. The trapezoidal integration scheme integrates displacements by applying equation (29) on the velocities stored at mesh points:

$$\mathbf{x}(t + \delta t) = \mathbf{x}(t) + 0.5(\mathbf{u}(\mathbf{x}(t)) + \mathbf{u}(\mathbf{x}(t + \delta t)))\delta t = \mathbf{x}(t) + \delta_t^{tr} \mathbf{x} \quad (164)$$

with second-order accuracy in time, where  $\delta_t^{tr} \mathbf{x}$  is the point displacement integrated with the trapezoidal integration rule. Note that the trapezoidal integration requires the velocity  $\mathbf{u}(\mathbf{x}(t + \delta t))$  to be known. Implicit schemes cause decoupling between the volume fraction advection equation and the rest of the two-phase **NS** equation system, when physical velocities are used.

Therefore, an alternative explicit temporal integration can be used that is based on Taylor series:

$$\mathbf{x}(t + \delta t) = \mathbf{x}(t) + \mathbf{u}(\mathbf{x}, t)\delta t + \frac{d\mathbf{u}}{dt}(\mathbf{x}, t)\delta t^2 + \mathcal{O}(\delta t^3) = \mathbf{x}(t) + \delta_t^T \mathbf{x} + \mathcal{O}(\delta t^3) \quad (165)$$

where  $\delta_t^T \mathbf{x}$  is the Taylor series based point displacement. Since  $\mathbf{u} = \mathbf{u}(\mathbf{x}, t)$ , from equation (165) it follows that

$$\delta_t^T \mathbf{x} = \mathbf{u}\delta t + (\partial_t \mathbf{u} + \nabla \mathbf{u} \cdot \mathbf{u})\delta t^2. \quad (166)$$

The temporal partial derivative term  $\partial_t \mathbf{u}$  can be discretized using any *backward* temporal discretization scheme, since the values are required in the *previous* time step:

$$\delta_t^T \mathbf{x} = \mathbf{u}\delta t + (\Delta_t \mathbf{u} + \nabla_c \mathbf{u} \cdot \mathbf{u})\delta t^2 + \mathcal{O}(\delta t^3) + \mathcal{O}_{\Delta t}(\delta t^{p+2}) + \mathcal{O}(\|\mathbf{d}\|^2), \quad (167)$$

where  $p$  is the largest truncation error exponent of the discrete differential temporal derivative operator  $\Delta_t$ . The consequence of  $\mathcal{O}_{\Delta t}(\delta t^{p+2})$  is the possibility to use first-order accurate Backward Euler (BE) derivative approximation for approximating  $\partial_t \mathbf{u}$  and still retain the overall error  $\mathcal{O}(\delta t^3)$ . The spatial approximation error is a result of approximating  $\nabla \mathbf{u}$  with a finite volume discrete differential operator  $\nabla_c$  that is second-order accurate on unstructured meshes. Note at this point that the use of the discrete gradient  $\nabla_c$  as outlined in chapter 3 is only possible with second-order accuracy *in space* if  $\delta_t^T \mathbf{x}$  is associated to the center of the finite volume (cell).

In practice, both  $\delta_t^{tr} \mathbf{x}$  and  $\delta_t^T \mathbf{x}$  are evaluated at cell centers and interpolated to cell corner points using either the IDW or the Taylor cell-point interpolation. Point displacements are evaluated at cell centers and interpolated onto mesh points (cell corner points) because the UFVFC scheme is to be coupled with a pseudo-staggered unstructured FV method applied to a single-field formulation of the NS system used for modeling two-phase flows, that stores velocities at cell centers. As already pointed out, Comminal et al. [26] have shown that spatial interpolation plays a significant role in the convergence order of a dimensionally un-split geometrical VoF method. For this reason, the use of an exact velocity field  $\mathbf{u}(\mathbf{x}, t)$  at cell corner points leads to an artificially high accuracy and convergence order, that may not be achievable when using a velocity which comes from the solution of the two-phase NS system.

### 6.3.2 Narrow band calculation

Geometrical operations required by the UFVFC scheme are computationally expensive. In order to increase the computational efficiency of the UFVFC scheme implementation, geometrical phase flux volume calculation  $V_f^\alpha$  is performed with a so-called *narrow band* of cells: a set of cells that are surrounding the PLIC interface.

The narrow band calculation is used when building the flux stencil  $N_{F_f}$  in algorithm 7. A flux stencil can be defined as a set of indices of all cells that intersect the face  $\mathbb{F}_f$ ,

$$N_{F_f} = \{c : \mathbb{C}_c \cap \mathbb{F}_f \neq \emptyset\}. \quad (168)$$

The cells that belong to the flux stencil are used to intersect the flux volume  $V_f$  in the process of computing the phase flux volume contributions  $V_{f,c}^\alpha$  as outlined in chapter 4. The larger the flux stencil, more intersections between cells and flux polyhedra need to be performed. Since the flux polyhedron (flux volume) is generated by sweeping the face backward along discrete Lagrangian trajectories, it will not intersect all cells in the flux stencil that is defined by equation (168).

A simple reduction of the  $N_{F_f}$  set can be done by relying on the axis-aligned bounding box **AABB** intersection test

$$N_{F_f} = \{c : \text{overlap}(AABB(\mathbb{C}_c), AABB(V_f))\}. \quad (169)$$

Equation (169) uses the geometry of the swept face in order to determine which cells of the  $N_{F_f}$  computed with equation (168) are actually contributing to the computation of the phase flux volume  $V_f^\alpha$ . This is done by constructing axis-aligned bounding boxes of each cell in  $N_{F_f}$  and checking if the **AABB** of the cell overlaps with the **AABB** of the flux volume  $V_f$ . The **AABB** overlap test used in equation (169) is a simple algorithm described by Ericson [39, chapter 4] or Schneider [121, section 11.12.6]. One detail related to equation (169) is very important to emphasize: the **AABB** is not built from  $V_f$ , it is built from the original and swept face points. The sets of original and swept face points used to construct  $V_f$  (cf. figure 4) contain the maximal and minimal points of the **AABB** of  $V_f$ . Therefore, constructing the flux volume only to simplify it again by representing it with its **AABB** would introduce unnecessary computation for all stencils where  $AABB(\mathbb{C}_c)$  and  $AABB(V_f)$  do not overlap  $\forall c \in N_{F_f}$ , since in this case  $V_f$  is not used to geometrically compute the phase flux volume contribution  $V_{f,c}^\alpha$ .

Once the reduced flux stencil is computed using equation (169), in the next step of the **UFVFC** algorithm 7,  $\mathbb{F}_f$  is categorized as a *narrow band face* if

$$\exists \mathcal{H}_{\alpha,c}, c \in N_{F_f} \text{ such that } \exists \mathbf{p} \in V_f : \mathbf{p} \notin \mathcal{H}_{\alpha,c}. \quad (170)$$

This condition is formulated algorithmically with algorithm 8 and the actual flux volumes  $V_f$  resulting from the application of algorithm 8 on a circular interface in a 2D shear flow are shown in section 6.3.2. Note that  $\mathbb{P}_\delta$  used in the implementation of equation (170) represent the moved mesh points (e.g.  $\mathbb{P} + \{\delta_t^T \mathbf{p}, \forall \mathbf{p} \in \mathbb{P}\}$ ). This is done in order to save computational costs by avoiding unnecessary distance calculation for points of the flux volume  $V_f$  that are a result of its triangulation, correction, or simplification operations. Obviously, the narrow band condition given by equation (170) and implemented with algorithm 8 results in a significant reduction in geometrical calculations. Section 6.3.2 shows clearly the flux volumes that are to be intersected to compute phase flux volume contributions. A consequence of checking for points that are in the negative half-space of the

interface element to enable geometrical intersections, is that those volumes that are completely immersed in phase 1 are handled as upwind volumes, as for them no intersection operations are necessary. Equation (170) represents only one possibility in reducing the computational complexity and increasing the accuracy of the **LE** flux-based geometrical **VoF** method, other complexity reduction algorithms are to be investigated in the future.

---

**Algorithm 8** Face in narrow-band **FNB**


---

```

1: for  $c \in N_{F_f}$  do
2:   Obtain half-space  $\mathcal{H}_{\alpha,c}$ .
3:   for  $p \in N_{fp}$  do
4:     Get face point  $\mathbf{p}_p \in \mathbb{P}$ .
5:     Get swept face point  $\mathbf{q}_p \in \mathbb{P}_\delta$ .
6:     if ( $\text{distance}(\mathcal{H}_{\alpha,c}, \mathbf{p}_p) < 0$ ) then
7:       return true
8:     end if
9:     if ( $\text{distance}(\mathcal{H}_{\alpha,c}, \mathbf{q}_p) < 0$ ) then
10:      return true
11:    end if
12:  end for
13: end for
14: return false

```

---

### 6.3.3 Flux polyhedron calculation

Geometrically computing the volume of a flux polyhedron includes a known problem of triangulating non-convex polyhedra with non-planar faces. Deciding if additional points (*Steiner points*) are required for the triangulation is an Non-deterministic Polynomial (**NP**)-hard problem (Rupert and Seidel [115]). Flux polyhedron calculation presented here is heuristic, it relies on the structure of a complex 3D polyhedron that is the result of sweeping the face along discrete Lagrangian trajectories and requires only a single additional Steiner point.

Section 6.3.2 shows that algorithm 8 and equation (170) are successful in reducing the number of flux volumes that are taking part in the geometrical calculation of the phase flux volume contribution  $V_{f,c}^\alpha$ . Next step of the **UFVFC** algorithm 7 is the actual computation of the flux volumes (flux polyhedra) shown in section 6.3.2. As covered in section 5.2, various ways of simplifying the flux volume shape in order to avoid complex geometrical operations that are required for 3D application of the geometrical **VoF** method were suggested by different researchers, leading to different **LE** flux-based geometrical **VoF** methods.

Calculation of the flux polyhedron (flux volume  $V_f$ ) is split in two steps: triangulation and correction for volume conservation. Both steps influence the stability, convergence and volume conservation of the geometrical **VoF** method. In section 4.1.3 the importance of both the triangulation and the correction for volume conservation for the **LE** flux-based geometrical **VoF** method is emphasized. Both computational steps are mutually dependent: error in the triangulation leads to an

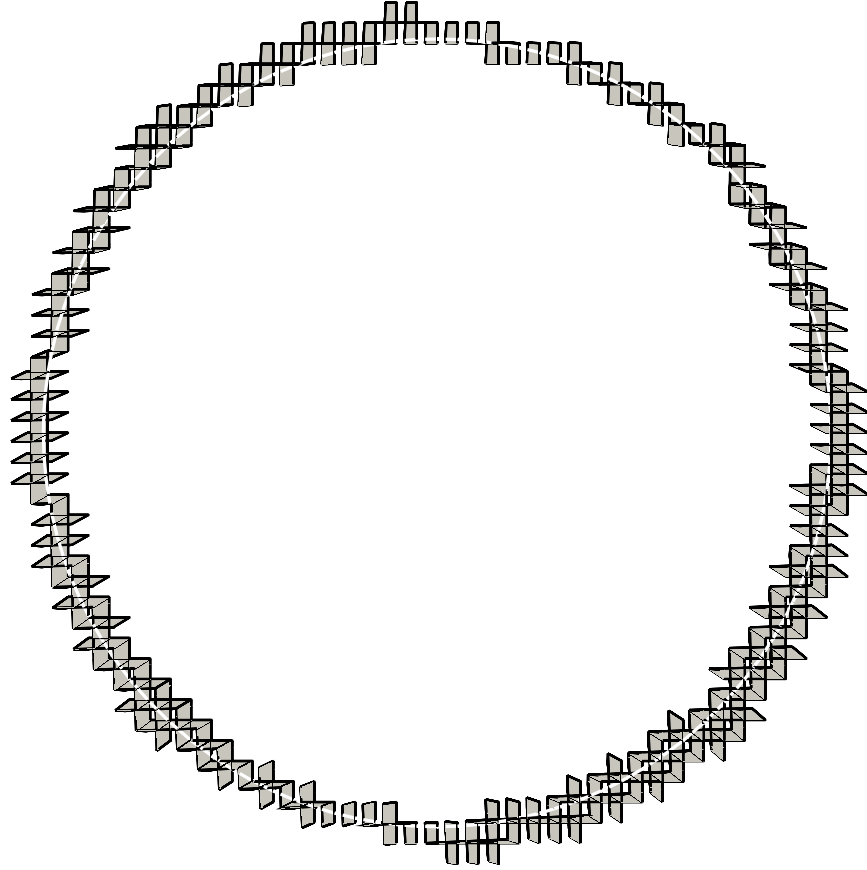
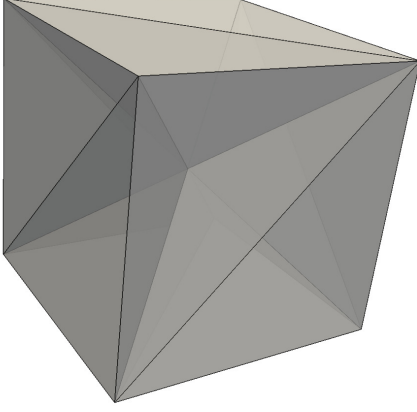


Figure 26: Narrow band flux volumes (gray volumes with black boundaries) and the **PLIC** interface elements (white lines), computed for a circular interface and the 2D shear velocity field from section 11.3, on an unstructured hexahedral mesh.

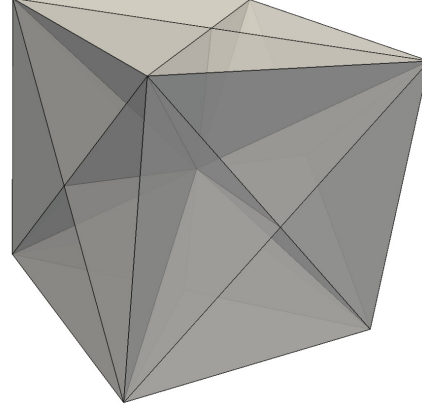
error in the volume calculation, which in turn leads to an error in the computation of the volume conservation error and, in the end, to an error in the correction for volume conservation. Consequently, it is of utmost importance that the volume bounded by face points and their swept counterparts are approximated accurately: an accurate flux polyhedron triangulation is necessary.

Those researchers that have not relied on simplifications of the flux volume that result in the approximation of the flux volume in the form of a convex polyhedron with planar faces, have suggested either the Oriented Triangulation (**OT**) (Owkes and Desjardins [99, figure 7b], Le Chenadec and Pitsch [71, figure 9b]) or the Barycentric Triangulation (**BT**) (Comminal et al. [26, figure 19 c], Jofre et al. [65, figure 6a]) for the flux polyhedron. However, already for the standard verification tests, configurations of point displacements appear that yield non-convex flux polyhedra that cannot be triangulated using either oriented or barycentric triangulation. For those displacement configurations an increase in the resolution of the triangulation, often mentioned in the literature as the solution for increasing the accuracy of the flux volume calculation, does not actually lead to a more accurate flux volume computation.

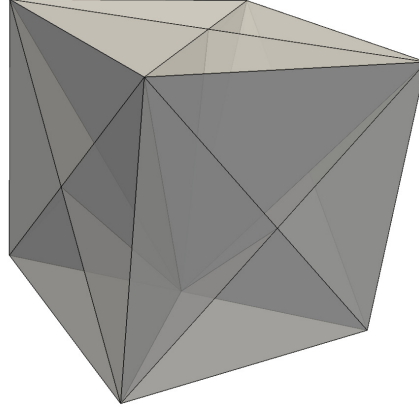
In order to clearly expose the problem of using oriented and barycentric triangulations in 3D, consider triangulations shown in figure 27. All triangulations



(a) Oriented triangulation of a cube created by vertically sweeping a square.



(b) Barycentric triangulation of a cube created by vertically sweeping a square.



(c) Flux-aware triangulation of a cube created by vertically sweeping a square.

Figure 27: Different triangulations of a cube.

result in a proper decomposition of a cube in tetrahedra, in the sense that the sum of the volume of all tetrahedra is equal to the volume of the cube. This is only possible because the triangulated cube does not have severely non-convex and non-planar faces, so the cube centroid lies within the cube. A version of the oriented triangulation shown in figure 27(a) uses the **OCPT** outlined in algorithm 9 to triangulate the faces of the cube and the centroid of the cube to decompose it into a set of tetrahedra.

---

**Algorithm 9** Oriented convex polygon triangulation **OCPT**

---

```

1:  $T = \emptyset$ 
2:  $\mathbf{p}_0$  is the first point in polygon  $\mathcal{Q}$ .
3: for  $p \in \{1..|\mathcal{Q}|\}$  do
4:    $T = T \cup \text{triangle}(\mathbf{p}_0, \mathbf{p}_p, \mathbf{p}_{\Pi_c(p+1)})$ 
5: end for
```

▷  $\mathcal{Q}$  is a set of points.

---

The barycentric triangulation relies on the polygon centroid (barycenter) to decompose convex polygons into sets of triangles, as described by algorithm 10 and shown in figure 27(b). The centroid of the cube is then used to further compute the set of tetrahedra.

**Algorithm 10** Barycentric convex polygon triangulation **BCPT**


---

```

1:  $T = \emptyset$ 
2:  $\mathbf{x}_c = \text{centroid}(\mathcal{Q})$ 
3: for  $p \in \{1..|\mathcal{Q}|\}$  do
4:    $T = T \cup \text{triangle}(\mathbf{x}_c, \mathbf{p}_p, \mathbf{p}_{\Pi_c(p+1)})$ 
5: end for

```

---

▷  $\mathcal{Q}$  is a set of points.

The oriented triangulation shown in figure 27(a) is not equivalent to the oriented triangulations used by Owkes and Desjardins [99], Le Chenadec and Pitsch [71] in the way the tetrahedra are constructed from triangulated faces. However, this does not affect the issue any oriented triangulation has in the accurate volume calculation for polyhedra with non-convex faces. Figure 28 illustrates the errors of both the oriented and the barycentric triangulation when applied on a flux volume generated for a test case that generates non-convex faces at two opposite sides of a flux volume.

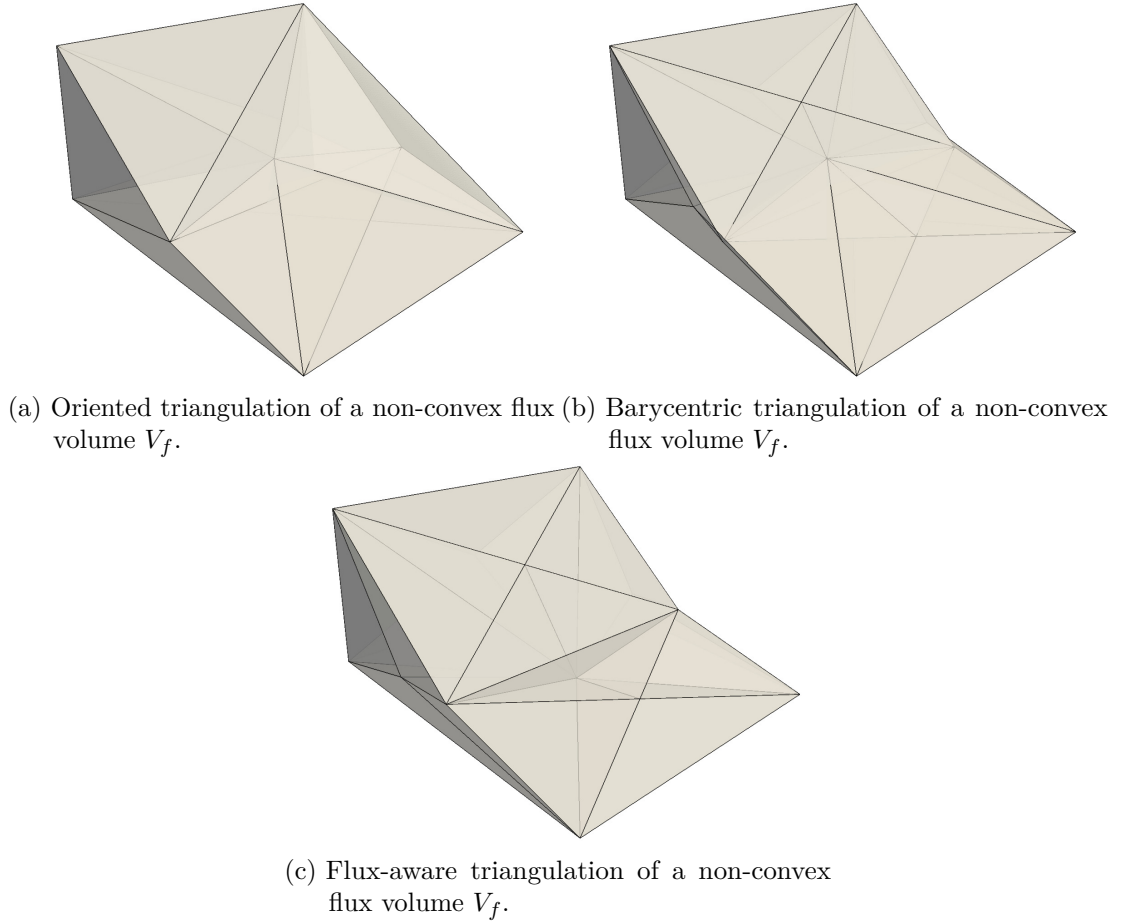


Figure 28: Different triangulations of a non-convex flux volume.

Figure 28(a) illustrates the error of the oriented triangulation: for the non-convex face on the right side of the flux volume, the oriented triangulation generates volume by triangulating the non-convex face in a wrong order, which results in a self-intersection. On the other hand, the barycentric triangulation has an error shown in figure 28(b). The error manifests itself in the form of a surplus volume



that is created as soon as the centroid of the triangulated shape is located outside outside of the shape. In this case a visible self-intersection happens, just like for the oriented triangulation. This error in generating volume by triangulation results in an error when computing the magnitude of the flux volume correction, leading in turn to increased stability, convergence and volume conservation errors.

The flux-aware triangulation whose result is shown in figure 28(c) avoids these issues completely. Flux-aware triangulation does not rely on the centroid of the flux volume to construct the tetrahedra and it employs a so-called *edge triangulation* for triangulating non-convex and non-planar faces: instead of using the centroid position, a position is found that triangulates strongly non-convex flux polyhedra as well. Edge triangulation used by the flux-aware triangulation for triangulating faces of the flux polyhedron makes sure that the area of each non-convex face is computed exactly for planar faces and approximated well for non-planar faces. Additionally, edge triangulation ensures conformity of triangulations for two adjacent flux polyhedra, so that no overlaps or holes can appear that would lead to either too much or too little phase flux volume to be calculated by the UFWFC scheme.

---

**Algorithm 11** Edge triangulation

---

```

1:  $\mathbf{D}_{02} = 0.5(\mathbf{p}_0 + \mathbf{p}_2)$  ▷ Compute the centers of the polygon diagonals.
2:  $\mathbf{D}_{13} = 0.5(\mathbf{p}_1 + \mathbf{p}_3)$ 
3:  $\mathbf{p}_0\mathbf{D}_{02} = \mathbf{D}_{02} - \mathbf{p}_0$  ▷ Connect the first edge point with diagonal centers.
4:  $\mathbf{p}_0\mathbf{D}_{13} = \mathbf{D}_{13} - \mathbf{p}_0$ 
5:  $\hat{\mathbf{E}}_{01} = \frac{\mathbf{p}_1 - \mathbf{p}_0}{\|\mathbf{p}_1 - \mathbf{p}_0\|}$  ▷ Compute the base edge unit vector.
6:  $w_{D02} = \frac{1.0}{\|\mathbf{p}_0\mathbf{D}_{02} - (\mathbf{p}_0\mathbf{D}_{02} \cdot \hat{\mathbf{E}}_{01})\hat{\mathbf{E}}_{01}\| + \epsilon_g}$  ▷ Inversed height weights.
7:  $w_{D13} = \frac{1.0}{\|\mathbf{p}_0\mathbf{D}_{13} - (\mathbf{p}_0\mathbf{D}_{13} \cdot \hat{\mathbf{E}}_{01})\hat{\mathbf{E}}_{01}\| + \epsilon_g}$ 
8:  $\mathbf{p}_T = \frac{(\mathbf{D}_{02}w_{D02}) + \mathbf{D}_{13}w_{D13}}{w_{D02} + w_{D13}}$  ▷ Compute the triangulation (Steiner) point.
9: Initialize triangulation  $T = \emptyset$ 
10: for  $p \in \{1..|\mathcal{Q}|\}$  do
11:    $T = T \cup \text{triangle}(\mathbf{p}_T, \mathbf{p}_p, \mathbf{p}_{\Pi_c(p+1)})$ 
12: end for

```

---

Edge triangulation is outlined by algorithm 11. It is not a general-purpose triangulation algorithm for non-convex polygons: it is based on the assumption that the polygon is created by sweeping the edge along the two discrete displacements of its end points. Therefore, edge triangulation represents a solution to the problem of triangulating non-planar and non-convex quadrilateral faces of the flux polyhedra generated by tracing the face along discrete Lagrangian trajectories. Note that every swept edge of an n-sided polygon is quadrilateral. The motivation behind using a specialized approach compared to general triangulation algorithms for non-convex polygons lies in its simplicity and resulting speed of execution as well as the possibility to handle non-convex polygons with overlapping edges.

Consider the non-convex polygon with an overlapping edge pair  $(\mathbf{p}_0\mathbf{p}_1, \mathbf{p}_3\mathbf{p}_0)$  shown in figure 29(a): in order to compute the volume of a flux polyhedron whose side is similar to this polygon, the flux-aware triangulation algorithm must be able



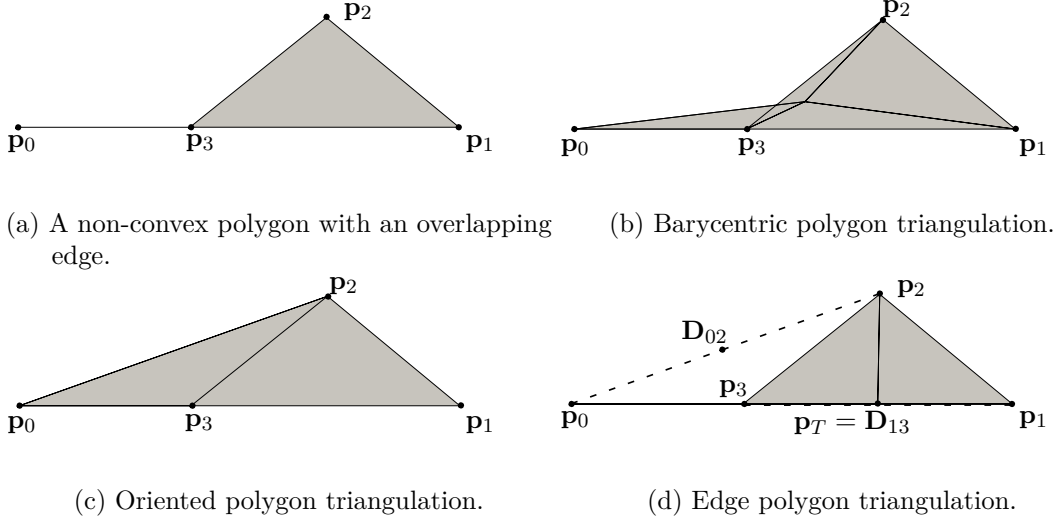


Figure 29: Different triangulations applied on a non-convex polygon.

to triangulate the polygon and compute the gray shaded area correctly. Figure 29(b) shows how the barycentric triangulation generates artificial area for this polygon, and the oriented triangulation has a similar effect shown in figure 29(c). Note that the oriented triangulation might work even for the polygon shown in figure 29(a), provided that the polygon is permuted until  $\mathbf{p}_1 = \mathbf{p}_0, \mathbf{p}_2 = \mathbf{p}_1, \dots$ , however, performing the permutation  $n$ -times, based on a test that recognizes a degenerated oriented triangulation of every side face of the flux polyhedron is inefficient.

The simple edge triangulation is an efficient and accurate triangulation, even for a non-convex polygon with overlapping edges as shown in figure 29(d). Algorithm 11 computes the midpoints of the two diagonals of the polygon ( $\mathbf{D}_{02}, \mathbf{D}_{13}$ ) and uses the inverse orthogonal distances to the edge ( $\mathbf{p}_0, \mathbf{p}_1$ ) as weights in order to compute the triangulation point  $\mathbf{p}_T$ . The triangulation point is also called the Steiner point: it is an additional point added to a polygon or a polyhedron that makes it possible to perform the triangulation correctly. More information on standard polygon and polyhedron triangulation methods and the role of Steiner points can be found in [32] and [25]. In the case of the polygon shown in figure 29(a), the midpoint of the diagonal  $\mathbf{D}_{13}$  lies on the edge  $\mathbf{p}_0\mathbf{p}_1$  and therefore has the orthogonal distance to the edge equal to 0. The inverse distance weight calculation is therefore stabilized in algorithm 11 using  $\epsilon_g = 1e - 15$ . As a result of the zero distance to the edge, the triangulation point  $\mathbf{p}_T$  of the polygon corresponds to the diagonal median point  $\mathbf{D}_{13}$  in this case. This leads to the computation of the shaded area in figure 29(d) that corresponds exactly to the shaded area in figure 29(a).

The images shown in figure 29 represent the result of a single parameter vector that belongs to the parameter space used to validate the edge triangulation by applying different configurations of the displacements ( $\mathbf{p}_0\mathbf{p}_3, \mathbf{p}_1\mathbf{p}_2$ ) that are generated with different length and angle increments with respect to the base edge  $\mathbf{p}_0\mathbf{p}_1$  of the polygon shown in figure 29(a). The case that does not involve an overlap, but an intersection of  $\mathbf{p}_2\mathbf{p}_3$  with  $\mathbf{p}_0\mathbf{p}_1$  cannot be handled by edge triangulation, instead an intersection-based separation of convex polygons must be used. Still, numerical tests in chapter 11 show that there is a significant positive

influence of the edge triangulation to the overall accuracy and convergence of the solution for the verification cases.

It is also important to emphasize that the edge triangulation described by algorithm 11 returns the same result as the barycentric triangulation for convex and planar polygons. Were the polygon  $(\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3)$  a square, the midpoints of its diagonals would coincide with the center of the square. Because of this, handling of non-convex polygons by triangulation is performed by the edge triangulation without explicitly handling the non-convexity as a special case in the algorithm. This supports one of the main advantages of using edge triangulation, besides delivering more accurate polygonal areas: its speed of execution.

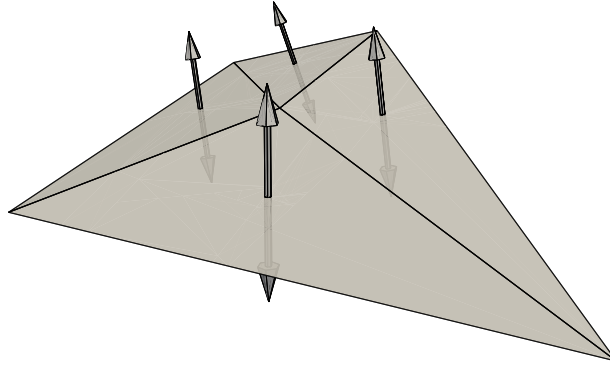


Figure 30: Continuity of the edge triangulation. Edge triangulation results in a unique definition of the triangulation point for two adjacent flux polyhedra, as well as the consistent orientation of the normal area vectors that belong to both triangulations, without requiring any special case handling for non-convexity.

Another important effect of using the inverse orthogonal distance weight factor for computing the triangulation point is the continuity of the triangulation for the side polygons of two adjacent flux polyhedra. This is shown for a single test configuration of edge point displacements in figure 30. Algorithm 11 will result in exactly the same set of triangles, irrespective of the orientation of the edge  $(\mathbf{p}_0, \mathbf{p}_1)$ , or  $(\mathbf{p}_1, \mathbf{p}_0)$ . Furthermore, the orientation of the triangle normal area vectors will be automatically oriented in the opposite direction for the adjacent flux polyhedron. This important property of the edge triangulation ensures that no special care must be taken when addressing the edge points and triangulating the side faces of the flux polyhedron, which further reduces the need for special case handling in the algorithm and increases its overall computational efficiency.

Once the side faces of the flux volume have been approximated by the edge triangulation, the set of tetrahedra is computed from all triangulated faces in order to decompose the flux polyhedron. The barycentric triangulation error shown in figure 28(b) can appear even when there is only a slight non-convexity of the flux polyhedron. Because of this, the centroid of the flux polyhedron cannot be used to compute a triangulation point of the flux polyhedron. In figure 28(c), the triangulation point of the flux-aware triangulation is computed as the centroid of

the bottom face of the cube. Algorithm 12 contains the steps of the flux-aware triangulation that lead to this result in the computation of the triangulation (Steiner) point.

Algorithm 12 projects the swept polygon onto the base polygon of the flux polyhedron and computes a set of intersection points: points of the projected face that are inside the base polygon and vice versa. The triangulation point is chosen as the arithmetic average of the set of intersection points. This approach for computing the triangulation point for the flux polyhedron is based on the way the flux polyhedron is generated: by sweeping the base polyhedron using a set of displacement vectors. Because the additional points of the flux polyhedron are created by the sweeping the base polygon, flux-aware triangulation assumes that there exists a triangulation point  $\mathbf{p}_T$  on the base polygon, such that  $(\mathbf{p}_T - \mathbf{p}_{0,T}) \cdot \mathbf{n}_T > 0$ , for every triangle  $T$  in every triangulation of every face of the flux volume. In other words, all triangles resulting from the edge triangulation of side faces and the barycentric triangulation of the swept (cap face) are *visible* to the triangulation point  $\mathbf{p}_T$  computed by the flux-aware triangulation. As a consequence, tetrahedra computed using such a triangulation point and the set of face triangles have no self intersections with each other, and their union is equal to the flux volume, even when strongly non-convex and non-planar faces are present.

---

**Algorithm 12** Flux-aware triangulation

---

- 1: Triangulate the swept (cap) face using barycentric triangulation.
  - 2: Triangulate side faces using edge triangulation.
  - 3: Initialize projected points  $\mathbb{P}'_{f,\delta} = \emptyset$
  - 4: **for**  $p \in \{1..|\mathbb{F}_f|\}$  **do** ▷ Project the cap points onto base plane.
  - 5:      $\mathbb{P}'_{f,\delta} \cup \mathbf{p}_{\delta,p} - (\mathbf{p}_{\delta,p} - \mathbf{p}_p) \cdot \hat{\mathbf{S}}_f \hat{\mathbf{S}}_f, \mathbf{p}_{\delta,p} \in \mathbb{P}_\delta, \mathbf{p}_p \in \mathbb{P}$
  - 6: **end for**
  - 7: Initialize intersection points  $\mathbb{P}_f^i = \emptyset$
  - 8: **for**  $p \in \{1..|\mathbb{P}'_{f,\delta}|\}$  **do** ▷ Intersect projected and base polygon.
  - 9:     **if**  $\|(\mathbf{p}'_p - \mathbf{p}_p) \times \hat{\mathbf{S}}_f\| > 0$  **then** ▷  $\mathbf{p}'_p$  is left of  $\mathbf{p}_p$ .
  - 10:          $\mathbb{P}_f^i = \mathbb{P}_f^i \cup \mathbf{p}'_p$
  - 11:     **end if**
  - 12:     **if**  $\|(\mathbf{p}_p - \mathbf{p}'_p) \times \hat{\mathbf{S}}_f\| > 0$  **then** ▷  $\mathbf{p}_p$  is left of  $\mathbf{p}'_p$ .
  - 13:          $\mathbb{P}_f^i = \mathbb{P}_f^i \cup \mathbf{p}_p$
  - 14:     **end if**
  - 15: **end for**
  - 16:  $\mathbf{p}_T = \frac{\text{sum}(\mathbb{P}_f^i)}{|\mathbb{P}_f^i|}$
  - 17: Build tetrahedra using  $\mathbf{p}_T$  and face triangulations.
- 

A computational detail of the flux-aware triangulation algorithm is important to mention: there is no need to triangulate the base polygon, as the triangulation point lies in the plane of the base polygon, so the resulting tetrahedra would have 0 volume. Flux-aware triangulation is a heuristic one: within the scope of this work, no theoretical proof of the validity of the assumption for the triangle visibility from the perspective of the triangulation point is done. Therefore, the accuracy of the flux-aware triangulation was measured for a set of parameterized polyhedra whose non-convexity was controlled by the parameter, and whose volume is easily

computed exactly. Polyhedra shown in figure 28 are exactly such candidates, and they represent a single set of parameters used to experimentally validate the flux-aware triangulation. Even though the accuracy of the flux-aware triangulation was tested for general non-convex polyhedra with non-planar faces only as the part of the **UFVFC** scheme, figure 31 shows the triangulation produced by algorithm 12 for a complex flux polyhedron that has strongly non-planar and non-convex faces.

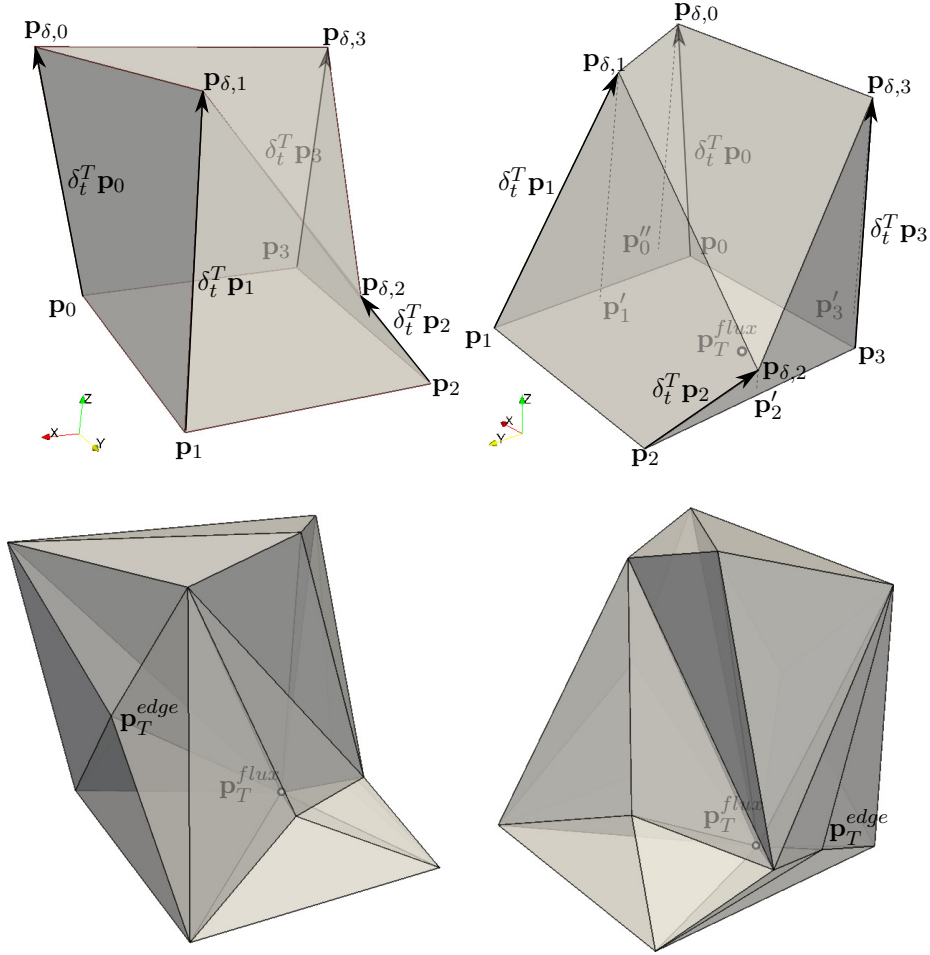


Figure 31: Flux-aware triangulation applied on a non-convex polyhedron with non-convex and non-planar faces. Upper image shows the rendering of the test flux polyhedron in a visualization software that relies on oriented triangulation for visualizing 3D polyhedra. Lower image shows the geometrical result of the flux-aware triangulation applied on the upper polyhedron. Base polygon points, their respective displacements, displaced points and resulting triangulation points follow the notation used in algorithm 12.

Results from the triangulation test shown in figure 31 are also shown in chapter 4, in figure 9 to introduce the reader unfamiliar with the geometrical **VoF** method with the impact a triangulation can have on the volume magnitude of a complex 3D flux polyhedron with non-planar and non-convex faces. In figure 31, notation from algorithm 12 is used to represent individual steps of the flux-aware triangulation algorithm, with the focus on obtaining the volume triangulation point. Triangulation

points of the side faces  $\mathbf{p}_T^{edge}$  of the flux polyhedron denoted in figure 31 are obtained using the above described edge triangulation.

Once the flux volume has been approximated using either flux, oriented or barycentric triangulation, its magnitude can be computed as a sum of volumes of all tetrahedra that build the triangulation. As outlined in section 4.1.1, the magnitude of the geometrically approximated flux volume will not be the same as the volume magnitude resulting from the integration of the volumetric flux over the discrete time step  $\delta t$ . This leads to a difference in the volume magnitude, between the geometrical and the integrated flux volume, a difference that must be corrected for in order to maintain volume conservation.

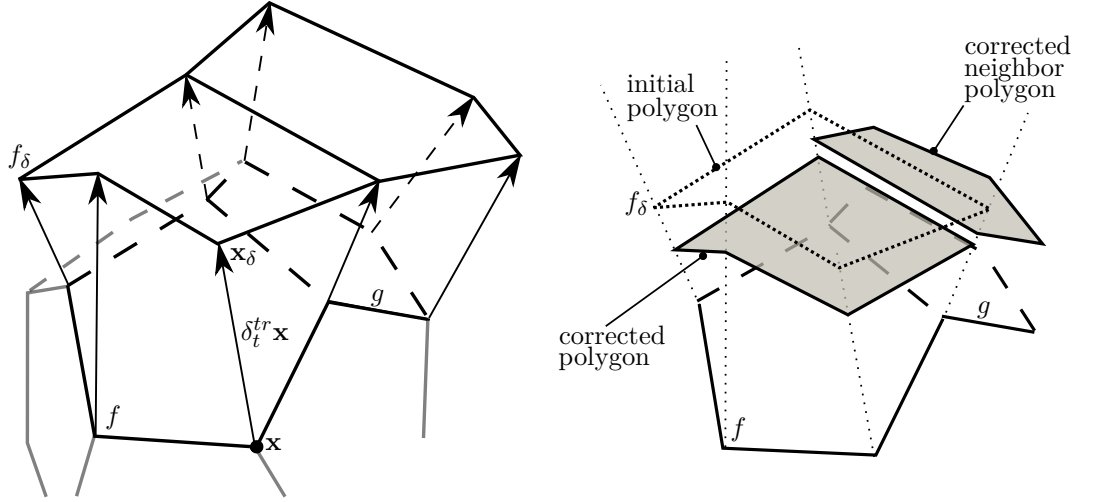
In [78], the initial implementation of the UFVFC scheme relied on an iterative correction of the flux polyhedron for volume conservation. Parallel to the further development of the UFVFC scheme, other researchers have used a single-step exact correction, based on the barycentric triangulation of the cap face of the flux volume (Jofre et al. [65], Owkes and Desjardins [99]). Section 6.3.3 contains schematic diagrams of different corrections for volume conservation as well as the schematic representation of the volume parametrization used by the pyramid correction.

Since the point displacements  $\delta_t^{tr} \mathbf{x}, \delta_t^T \mathbf{x}$  are interpolated from cell centers to cell corner points as described in section 6.3.1, even if the flux-aware triangulation accurately approximates the volume of the flux polyhedron, its magnitude will not correspond to the integral of the volumetric flux over the time step  $\delta t$  given by

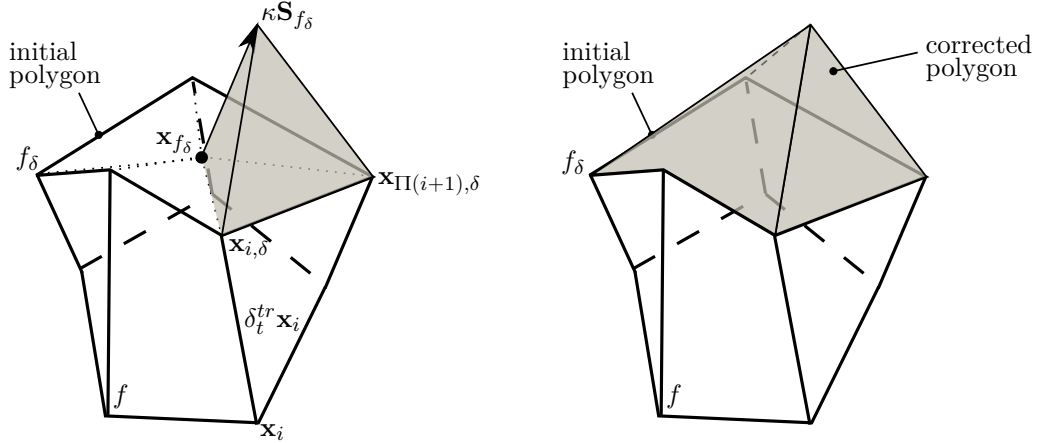
$$V_f = \int_t^{\tau+\delta\tau} F_f(t) dt, \quad (171)$$

where  $F_f = \mathbf{u}_f(t) \cdot \mathbf{S}_f$  is the volumetric flux over the face  $f$ , if the unstructured FV method is used for equation discretization. In chapter 4, equation (61) integrates equation (171) using the rectangle quadrature rule with first-order accuracy, to ensure a simple introduction of the geometrical VoF method. Researchers have been using different integration schemes: López et al. [75] have used the trapezoidal quadrature rule, Jofre et al. [65] are using the rectangle quadrature rule, Comminal et al. [26] are using a 4th-order Runge-Kutta method for computing the  $V_f$  using Lagrangian backward tracking with trilinear interpolation for the vertex velocities, etc.

The UFVFC scheme uses the trapezoidal quadrature rule and the trapezoidal quadrature rule with Taylor series predictor step to integrate equation (171), similar to the point displacement integration of  $\delta_t^{tr} \mathbf{x}, \delta_t^T \mathbf{x}$  given by equations (164) and (165). Since  $F_f = F_f(t)$ , the Taylor series based quadrature rule is different from equation (165), in the sense that no material derivative is required, so the discrete differential operator of the flux does not have the spatial term. The trapezoidal flux integration quadrature with Taylor series predictor is called *Taylor*



(a) Initial flux polyhedra created by tracing backwards the points  $\mathbf{x}$  along their respective displacements  $\delta_t^{tr} \mathbf{x}$  for two faces:  $f$  and  $g$ . (b) Iterative correction of two flux polyhedra from faces  $f$  and  $g$  may lead to an overlap or an offset between adjacent flux polyhedra.



(c) Pyramidal correction is based on swept face triangulation and normal parameterization. (d) Final cap face resulting from the pyramidal correction is a triangulation.

Figure 32: Iterative and pyramid correction of for volume conservation.

*flux integration* from this point on in the text. The Taylor flux integration simply extrapolates the flux in the new time step using Taylor series

$$\begin{aligned}
 V_f &= 0.5(F_f^{n+1} + F_f^n)\delta t + \epsilon_{tr} \\
 &= 0.5(2F_f^n + \partial_t F_f^n \delta t)\delta t + \epsilon_{tr} + \mathcal{O}_T(\delta t^3) \\
 &= 0.5(2F_f^n + \Delta_t F_f^n \delta t)\delta t + \epsilon_{tr} + \mathcal{O}_T(\delta t^3) + \mathcal{O}_{\Delta_t}(\delta t^{p+2}),
 \end{aligned} \tag{172}$$

where  $\epsilon_{tr}$  is the trapezoidal quadrature error over  $\delta t$  (equation (30)),  $\mathcal{O}_T$  is the truncation error of the Taylor series and  $\mathcal{O}_{\Delta_t}$  is the error of the discrete temporal difference operator on  $F_f$  at the old time step. Note that  $n$  and  $n+1$  are used

to denote current and next time step, following equation (28). The  $p + 2$  in the error term  $\mathcal{O}_{\Delta t}$  comes from multiplication with  $\delta t^2$  and consequently allows simple backward Euler temporal differencing to be used for the differential operator  $\Delta_t$ :

$$V_f^T = 0.5(2F_f^n - F_f^{n-1})\delta t + \mathcal{O}(\delta t^3). \quad (173)$$

This flux integration scheme requires only a single old field value to be stored in order to maintain third order accuracy. Just as the Taylor displacement integration, this scheme is explicit in the sense that the values at the next time step  $n + 1$  do not have to be known, a fact that reinforces the coupling of the volume fraction equation with the rest of the NS equation system for two-phase flows.

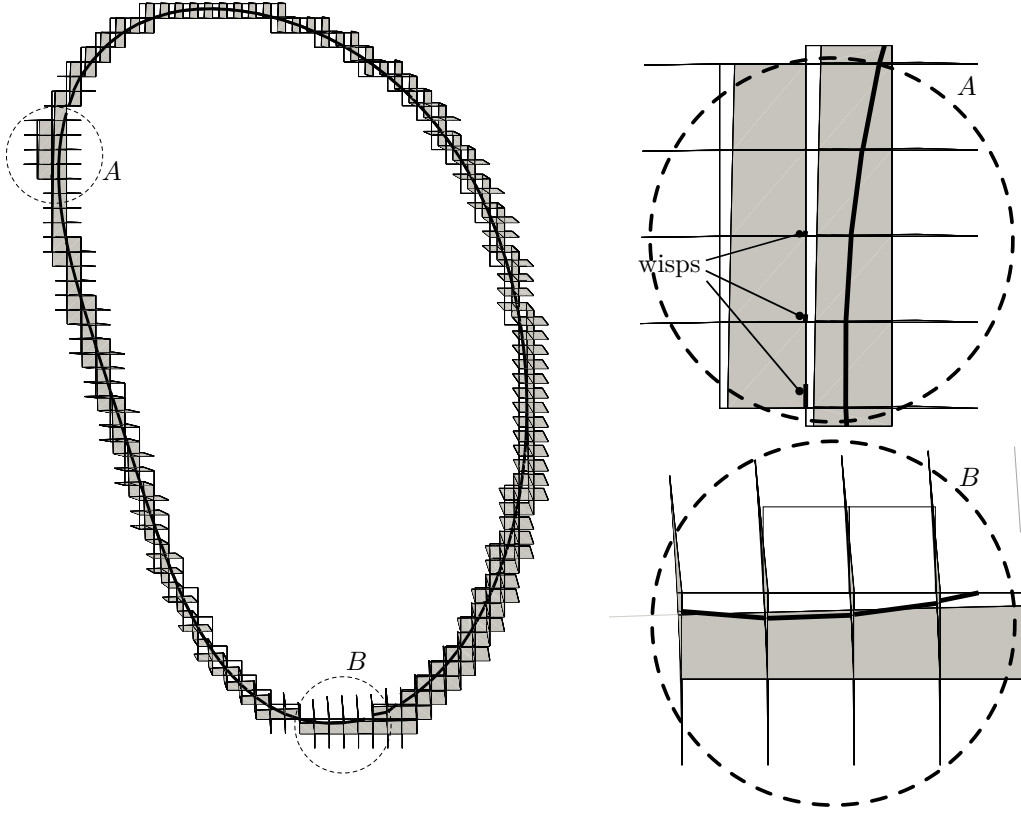
Once the flux volume magnitude has been integrated with at least second-order accuracy, the correction of the flux polyhedron must adjust the flux polyhedron so that its volume magnitude corresponds to the volume magnitude given by the numerical flux integration. The iterative correction for volume conservation shown schematically in figure 32(b) of the flux polyhedron shown schematically in figure 32(a) adjusts point displacements  $\delta_t^{tr} \mathbf{x}$  of face  $f$  by multiplying them with a scalar factor  $c\delta_t^{tr} \mathbf{x}$  and iteratively modifying the scalar factor until the volume of the triangulation of the flux polyhedron is equal to the volume given by the numerically integrated volumetric flux. As shown in figure 32(b), and also pointed out by Jofre et al. [65], Comminal et al. [26], Owkes and Desjardins [99], the iterative correction can result in an overlap between two neighboring flux polyhedra. Furthermore, the verification tests that include a spatially and temporally varying explicit velocity field in section 11.3 have confirmed the statement by Jofre et al. [65] that the iterative correction is not as efficient as the exact pyramidal correction.

Iterative correction was proposed first by López et al. [75] (using the Brent's root finding method), and they have commented on the fact that for large CFL values the flux polyhedra can exceed the bounds of the flux stencil  $N_{F_f}$ .

Figure 33 contains the geometry of the thin flux polyhedra for the 2D shear verification case described in section 11.3. In parts of the domain marked with A and B (shown in figure 33(b)) the flux polyhedra become very thin and difficult to correct for volume conservation. Especially in the B region, with  $CFL = 1$  the flux volumes become approximately of the size of the computational cell: in figure 33(b), the cells are shown so that a visual size comparison can be done against the flux volumes. Besides being computationally more expensive, the scaled flux correction also introduces accuracy issues, as the incorrect handling of very thin flux polyhedra can cause the UVFVC scheme to generate so-called *wisps*: small artificial interface elements that appear near the PLIC interface as shown in figure 33(b). It is important to emphasize that the test case shown in figure 33 was computed using the scaled correction and the reconstruction tolerance  $\epsilon_r = 1e - 09$ , which means that the volume fraction levels already at  $\alpha_c > 1e - 09$  are handled with PLIC reconstruction.

An exact correction (pyramid flux correction) was proposed by Jofre et al. [65, equation 20] and Owkes and Desjardins [99, equation 28] and it is shown in figures 32(c) and 32(d). The pyramid correction described by Jofre et al. [65], Owkes and Desjardins [99] can be generalized for n-sided polygons and even further





(a) Thin iteratively corrected flux polyhedra of (b) Thin flux polyhedra exceeding the cell size the 2D shear verification case.

Figure 33: Problematic thin flux polyhedra generated for the 2D shear verification case by the iterative flux correction for volume conservation. With  $CFL = 1$ , the size of the flux polyhedra becomes comparable to the cell size.

simplified. Consider the shaded tetrahedron volume shown in figure 32(c). The pyramidal correction of the flux volume  $V_f$  can be defined based on the swept face centroid displacement  $\mathbf{x}_{f,\delta}$  along the direction  $\kappa \mathbf{S}_{f_\delta}$ ,  $\kappa \in \mathbb{R}$ . In this case, the volume of a single tetrahedron shown in figure 32(c) is defined with

$$V_{T,i} = \frac{1}{6} \kappa \mathbf{S}_{f_\delta} \cdot ((\mathbf{x}_{i,\delta} - \mathbf{x}_{f_\delta}) \times (\mathbf{x}_{\Pi_c(i+1),\delta} - \mathbf{x}_{f_\delta})). \quad (174)$$

Naturally, the volume enclosed by the initial polygon (cap face)  $f_\delta$  from below and the pyramidal corrected polygon in figure 32(d) from above is computed as

$$\begin{aligned} \delta V &= V_f^i - V_f^g = \sum_{i=1}^{|\mathbb{F}_f|} V_{T,i} \\ &= \sum_{i=1}^{|\mathbb{F}_f|} \frac{1}{6} \kappa \mathbf{S}_{f_\delta} \cdot ((\mathbf{x}_{i,\delta} - \mathbf{x}_{f_\delta}) \times (\mathbf{x}_{\Pi_c(i+1),\delta} - \mathbf{x}_{f_\delta})) \\ &= \frac{1}{3} \kappa \mathbf{S}_{f_\delta} \cdot \sum_{i=1}^{|\mathbb{F}_f|} \frac{1}{2} (\mathbf{x}_{i,\delta} - \mathbf{x}_{f_\delta}) \times (\mathbf{x}_{\Pi_c(i+1),\delta} - \mathbf{x}_{f_\delta}), \end{aligned} \quad (175)$$



where  $\delta V$  is the flux volume correction applied to the flux polyhedron for volume conservation, defined as the difference in the flux volume given by integrating equation (171)  $V_f^i$  and the geometrical flux volume given by triangulating the flux polyhedron  $V_f^g$ . If the normal area vector of the swept face  $f_\delta$  is computed using the barycentric triangulation (an assumption used in this text as well as by Jofre et al. [65], Owkes and Desjardins [99]), the sum term in the equation for  $\delta V$  is equal to the normal area vector of the face  $f_\delta$ . This significantly simplifies the computation of the volume correction parameter  $\kappa$ :

$$\delta V = \frac{1}{3} \kappa \|\mathbf{S}_{f_\delta}\|^2 \Rightarrow \kappa = \frac{3\delta V}{\|\mathbf{S}_{f_\delta}\|^2}. \quad (176)$$

If any direction vector  $\mathbf{a}$  is used to correct for volume conservation using the pyramid correction, the correction parameter  $\kappa$  can be computed as

$$\kappa = \frac{3\delta V}{\mathbf{a}_{f_\delta} \cdot \mathbf{S}_{f_\delta}}. \quad (177)$$

In flow regions with large shear rates, thin flux polyhedra similar to those shown in figure 33(b) appear. Correcting those flux polyhedra using the direction of the swept face normal leads to self-intersection of the resulting polyhedra within the flux polyhedron and not necessarily to self-intersection of the corrected flux polyhedron with the face  $\mathbb{F}_f$ . A simple way of reducing the amount of self-intersections for thin flux polyhedra is to modify the pyramid flux correction by using an average Lagrangian backward tracing direction given as

$$\mathbf{a}_{f_\delta} = \frac{1}{|\mathbb{F}_f|} \sum_{i=1}^{|\mathbb{F}_f|} \delta_t^{tr} \mathbf{x}_i, \quad (178)$$

It is important to note that  $\frac{\mathbf{a}_{f_\delta}}{\|\mathbf{a}_{f_\delta}\|} = \hat{\mathbf{S}}_f$  if  $\delta_t^{tr} \mathbf{x}_i$  are orthogonal to the face  $\mathbb{F}_f$ . Also, a negative  $\delta V$  leads to the negative value of  $\kappa$  which orients  $\mathbf{a}_{f_\delta}$  inwards as a reduction of  $V_f^i$  is required. This however does not change the outward orientation of the face area normal vectors of the triangles in the corrected polygon.

At this point in algorithm 7, the flux polyhedra have been corrected for volume conservation, so the calculation of the phase flux volume contributions  $V_{f,c}^\alpha$  that build the total phase flux volume  $V_f^\alpha$ , can be done.

#### 6.3.4 Phase flux volume calculation

Contributions to the total phase flux volume  $V_f^\alpha$  transported across the face  $\mathbb{F}_f$  are computed by the phase flux volume calculation algorithm outlined in algorithm 13. Calculation of the phase flux volume contributions is defined with equation (69) as

$$V_{f,c}^\alpha = \mathcal{H}_c \cap V_{f,c} = \mathcal{H}_c \cap (V_f \cap V_c).$$

---

**Algorithm 13** Phase flux volume calculation

---

```

1:  $V_f^\alpha = 0$ 
2:  $T_\delta = \text{triangulate}(\mathbb{F}_\delta)$ 
3: for  $c \in N_{F_f}$  do
4:   if  $\alpha_c > \epsilon_r$  then
5:     Build cell half-spaces  $H_c = \{\mathcal{H}_f : \mathcal{H}_f = (\mathbf{x}_f, \tilde{\mathbf{S}}_f), \tilde{\mathbf{S}}_f \text{ is oriented inward.}\}$ 
6:     Initialize intersection half-spaces  $H_c^i = \emptyset$ .
7:     for  $\mathcal{H}_i \in H_c$  do ▷ Collect cell half-spaces that intersect  $T_\delta$ .
8:       for  $\mathcal{T}_j \in T_\delta$  do
9:         if  $\text{do intersect}(\mathcal{H}_i, \mathcal{T}_j)$  then
10:           $H_c^i = H_c^i \cup \mathcal{H}_i$ 
11:          break
12:        end if
13:      end for
14:    end for
15:    if  $|H_c^i| > 0$  then ▷ Compute phase flux volume contribution.
16:       $V_{f,c}^\alpha = V_f^\alpha$  ▷  $V_f^\alpha$  is a set of tetrahedra.
17:      for  $\mathcal{H}_i \in H_c^i$  do
18:         $V_{f,c}^\alpha = V_{f,c}^\alpha \cap \mathcal{H}_i$  ▷ Intersect with a cell half-space.
19:      end for
20:      if  $\alpha_c < (1 - \epsilon_r)$  then
21:         $V_{f,c}^\alpha = V_{f,c}^\alpha \cap \mathcal{H}_{\alpha,c}$  ▷ Intersect with the PLIC half-space.
22:      end if
23:       $V_f^\alpha = V_f^\alpha + \text{volume}(V_{f,c}^\alpha)$  ▷ Update total phase flux volume.
24:    end if
25:  end if
26: end for

```

---

The **UFVFC** scheme reformulates the intersection  $V_f \cap V_c$  in algorithm 13 so that the number of half-space / polyhedron intersections required to compute this intersection becomes minimal. Instead of intersecting  $V_f$  with all the half-spaces of a cell  $c$ , *intersection half-spaces* are only used for intersection. An intersection halfspace  $\mathcal{H}_c^i$  is defined as the face  $\mathbb{F}_f$  of the cell  $\mathbb{C}_c$  with an *inward facing* area normal vector  $\tilde{\mathbf{S}}_f$  that intersects the triangulation of the swept face  $\mathbb{F}_\delta$ . Using only those halfspaces of the cell  $\mathbb{C}_c$  that actually do intersect the flux polyhedron  $V_f$  increases the computational efficiency of the geometrical **VoF** method substantially, because profiling across all the verification tests in chapter 11 shows that the phase flux volume calculation consumes most of the computational time.

The idea behind the definition of the intersection half-space can be understood by observing figure 34 which contains a single cell  $\mathbb{C}_c$  taken from the detail  $B$ , shown in figure 33(b). In order to compute the phase flux volume contribution (darker triangle) from the cell  $c$  ( $V_{f,c}^\alpha$ ), the intersection  $\mathbb{C}_c \cap V_f$  is reduced to an intersection with a single half-space  $\mathcal{H}_c^i$ , because only a single face of the cell  $\mathbb{C}_c$  intersects  $V_f^\alpha$  so that the intersection result has volume greater than zero. In a three-dimensional verification case, for the cell  $\mathbb{C}_c$  this results in a speedup by a factor of 6 in the serial algorithm computation, since the hexahedral cell  $\mathbb{C}_c$  has 6 faces.

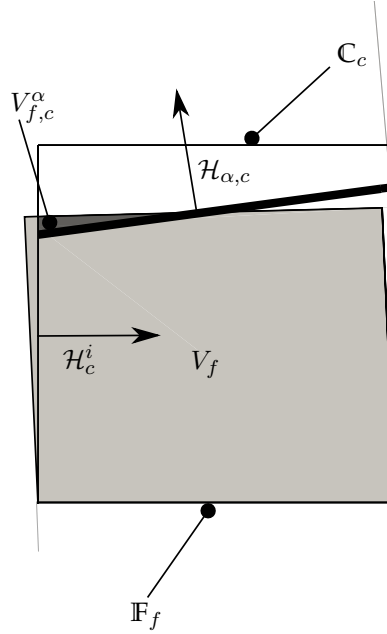


Figure 34: A cell  $\mathbb{C}_c$  with a single intersection half-space  $\mathcal{H}_c^i$  with the flux volume  $V_f$ .

Once the phase flux volume  $V_f^\alpha$  is computed using algorithm 13, equation (71) is used as the final step in the geometrical approximation of the volume fraction advection equation:

$$\alpha_c^{n+1} = \alpha_c^n - \frac{1}{V_c} \sum_f V_f^\alpha. \quad (179)$$

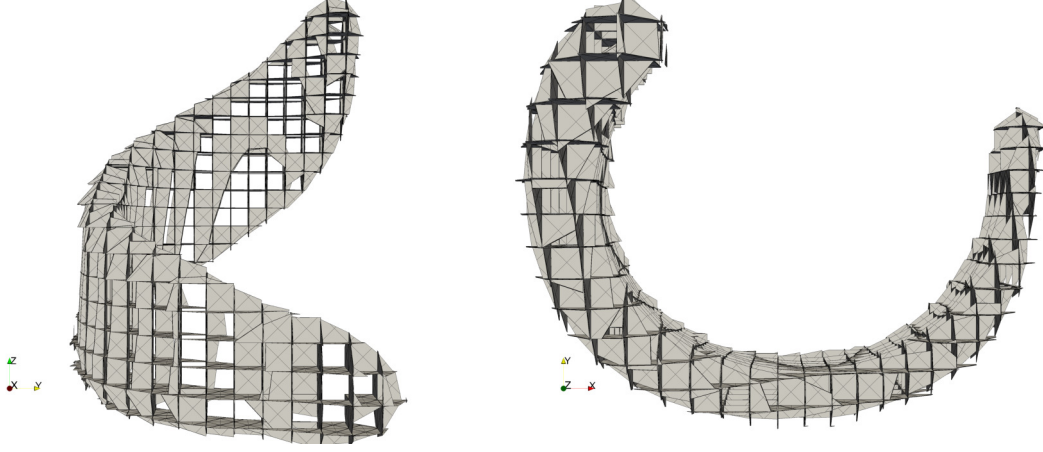


Figure 35: Phase flux volumes computed for the 3D shear verification case of Liovic et al. [73] ( $t = 1.5s$ ,  $N = 32$ ,  $CFL = 0.5$ ) from two perspectives.

In figure 35 the phase flux volumes computed with algorithm 13 are shown for the 3D shear case as proposed by Liovic et al. [73, figure 10]. The verification case proposed by Liovic et al. [73] tests the phase flux volume calculation for thin flux polyhedra in the similar way as for the 2D shear verification case shown in figure 33. Visualizing the interim results of geometrical operations is just as important as visualizing the PLIC interface to isolate those geometrical operations used for the phase flux volume calculation whose errors are not covered by a predefined set of tests for accuracy, robustness and topologically correctness.

### 6.3.5 Parallel implementation

Parallel implementation of the UFVFC scheme is based on the domain decomposition and message passing method using Open MPI. Computing both the inflow and outflow phase flux volumes on process boundaries would require knowledge of the mesh and the PLIC interface to be exchanged between processes. Since the PLIC interface is evolving, its geometrical data would in this case need to be exchanged across process boundaries every time step. When local dynamic AMR is used the topology of the mesh may change at the process boundary at every time step, if the PLIC interface is evolving across it. Opposite to this approach, for the UFVFC scheme, only outflow phase flux volumes are computed and exchanged across process boundaries, because the outflow from one process domain is equivalent to the inflow into the other. This minimizes the required number of exchanged messages.

It is often stated that the explicit nature of the geometrical VoF methods makes it possible to extend them easily for parallel computations. That is true in the sense that some parallel implementation can be programmed using the same computations as applied in the internal parts of the parallel process. However, from the aspect of high performance computing, geometrical VoF methods are algorithms that are heterogeneous in time and space: the computation is concentrated near the fluid interface that evolves in time and space. In order to achieve good parallel scaling

and efficiency properties for larger problems, load balancing is required (Jofre et al. [66]). More work is required to further exploit the explicit and heterogeneous nature of the geometrical **VoF** method on modern **HPC** architectures.

## 6.4 GEOMETRICAL OPERATIONS

Geometrical operations and models that are implemented by the generic geometrical C++ library described in section 6.6 and used by the **UFVFC** scheme are shown in table 2. Since the **UFVFC** scheme requires geometrical computations with real numbers, its accuracy will depend on the fixed precision floating point representation used by the computer architecture it is running on. For those algorithms in table 2 whose computation results with a logical true or false value (so-called *geometric predicates*), round-off error may lead to a wrong logical result. For example, if a wrong logical result is used to decide if a point is coplanar with a polygon within a geometrical intersection algorithm, the result might be a degenerate polyhedron with a face that has the area magnitude near machine tolerance  $\epsilon_m$ , in the case of an intersection between a cube and a half-space that is touching one of the cube points. Computing the surface area normal vector from such a degenerated polygon using equation (22) from chapter 3 may lead to an error caused by division by 0.

Therefore, there are two sources of errors in geometrical calculations that involve fixed precision floating point arithmetic: numerical and topological errors. Different methods were developed for avoiding numerical and topological errors and an overview of modern approaches is given by Li et al. [72]. Two main categories of approaches in resolving numerical and topological errors in geometric calculations have been developed so far: the arithmetic and the geometric approach. The arithmetic approach tries to solve both the numerical and the topological errors by concentrating on either improving the accuracy of arithmetical operations by enforcing exact arithmetic operations, or increasing robustness of the approximative arithmetic by adding bounds. The geometric approach tries to ensure valid geometric properties of the result within a geometrical algorithm. Li et al. [72] states what is confirmed by examining modern geometrical software libraries such as Boost Geometry Library (**BGL**) and Computational Geometry Algorithms Library (**CGAL**): the Exact Geometric Computation (**EGC**) geometric approach has been widely accepted for obtaining both topologically and numerically correct results from geometrical computations. The **EGC** approach ensures topological and not necessarily numerical exactness. However, the **EGC** approach is computationally expensive for the applications required by a geometrical **VoF** method, even when its efficiency is increased by relying on adaptive floating point precision resolution filters [72].

A literature survey given by Li et al. [72] points further to many contributions to the field of Computational Geometry that propose one or the other method for handling topological and numerical errors. Fabri et al. [40] emphasized that the computational efficiency is just as important as robustness and topological correctness, and this is especially the case for the geometrical **VoF** method because of the large amount of intersection and triangulation operations.

ALGORITHMS	MODELS
area	aabbbox
centroid	halfspace
displace	halfspace_sequence
distance_sqr	line_segment
equal_under_tolerance	line_intersection
extreme_projected_points	point_sequence_intersection
intersect	polygon_sequence_intersection
is_inside	triangle_intersection
orient	triangulation_intersection
triangulate	array_triangle
build	point_sequence
correct_volume	array_tetrahedron
distance	mixed_polyhedron
edge_triangulate	polygon_sequence
extreme_points	tetrahedron_sequence
flux_triangulate	triangle_sequence
normal	
perturb	
volume	
aabb_do_intersect	
do_intersect	

Table 2: Geometrical algorithms and models.

Requirements for efficiency, topological correctness, robustness and accuracy of the geometrical operations used by the **UFVFC** scheme are prescribed by the algorithm 7 and its sub-algorithms. Geometrical algorithms that are critical for the **UFVFC** scheme are: triangulation (**triangulate**), intersection (**intersect**) and the Axis-Aligned Bounding Box (**AABB**) overlap test (**aabb\_do\_intersect**). The triangulation is required for an accurate volume calculation, intersection for calculating the phase flux volume contributions and the **AABB** overlap test is the key algorithm in reducing the number of intersections to improve computational efficiency.

The proposed implementation of the algorithms outlined in table 2 relies on tolerance-based filtering when dealing with topologically complex cases. Topological correctness of geometrical operations is considered exclusively within requirements given by the geometrical **VoF** method. Algorithm 7 defines a geometrical model topologically if it can be intersected and its volume can be accurately computed (up to predictable tolerance), without causing the program to crash (robustness).

An example of a topologically challenging intersection with fixed precision floating point arithmetic between different half-spaces and a polygon are shown in

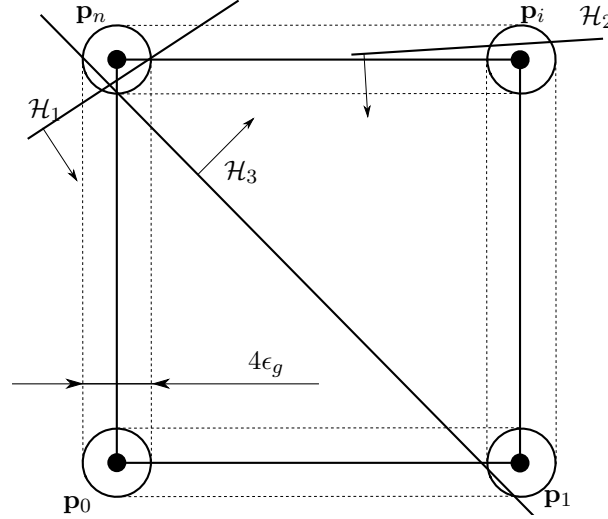


Figure 36: Topologically challenging intersection configurations for a polygon and a half-space resolved by tolerance-based logic.

figure 36. Fixed precision floating point arithmetic may cause different topological inconsistencies in the intersection results. The following considerations describe the topological errors that occur without the use of filtering: intersections with the polygon thickened by  $4\epsilon_g$  in figure 36 are not specially handled. In this case, the intersection with the half-space  $\mathcal{H}_1$  could create two intersection points that are indistinguishable from each other to the computer, or result in no intersection points whatsoever if all points are considered to lie on the half-space. Intersecting the polygon with the  $\mathcal{H}_2$  might not collect two intersection points, which would leave a polyhedron intersection without a cap polygon. If not handled properly, intersecting with the half-space  $\mathcal{H}_3$  can result with zero length edges  $(\mathbf{p}_1, \mathbf{p}_1)$  and  $(\mathbf{p}_n, \mathbf{p}_n)$ .

In all cases where zero length edges or zero area polygons are generated, the requirement of the geometrical VoF method for an accurate volume and area calculation will be fulfilled up to the round-off error given by fixed precision floating point arithmetic. The problem, however, are successive intersections required for phase flux volume calculations outlined in section 6.3.4: it is not possible to consistently intersect a polygon whose area is  $\approx 0$ , or that has edges with lengths  $\approx 0$ , without introducing either tests for degenerate shapes or avoiding divisions by zero by stabilizing divisions with  $\epsilon_g$ . Introducing tests for topologically inconsistent shapes creates a very large and unnecessary computational overhead, and stabilizing divisions by zero using  $\epsilon_g$  introduces errors that are larger than machine precision, and also difficult to estimate. Therefore, geometrical algorithms from table 2 that construct geometrical models are programmed using tolerance filters so that for a user-prescribed tolerance, a topologically consistent model is always generated. An intersection between a half-space and a polygon that is schematically shown in figure 36 is described by algorithm 14.

Algorithm 14 relies on the signed distance between a point and a half-space to determine if the point lies within  $4\epsilon_g$  next to the half-space. If this is the case,

**Algorithm 14** Polygon  $\mathcal{Q}$  / half-space  $\mathcal{H}$  intersection

---

```

1: Initialize intersection result  $R = \emptyset$ .
2: Initialize intersection points  $P = \emptyset$ .
3: for edges  $\mathbf{e} \in \mathcal{Q}$  do
4:    $\phi_0 = \text{distance}(\mathbf{x}_{e,0}, \mathcal{H})$ 
5:   if  $\phi_0 > -2\epsilon_g$  then ▷ Point is inside the half-space.
6:      $R = R \cup \mathbf{x}_{e,0}$ 
7:     if  $\phi_0 < 2\epsilon_g$  then ▷ Point is an intersection point.
8:        $P = P \cup \mathbf{x}_{e,1}$ 
9:     end if
10:  end if
11:   $\phi_1 = \text{distance}(\mathbf{x}_{e,1}, \mathcal{H})$ 
12:  if  $\phi_0\phi_1 < 0$  and  $|\phi_0| > 2\epsilon_g$  and  $|\phi_1| > 2\epsilon_g$  then ▷ Regular intersection.
13:     $\mathbf{p}_i = \text{INTERSECT}(\mathbf{x}_{e,0}, \mathbf{x}_{e,1}, \mathcal{H})$  ▷ Compute intersection point  $\mathbf{p}_i$ .
14:     $R = R \cup \mathbf{p}_i$ 
15:     $P = P \cup \mathbf{p}_i$ 
16:  end if
17: end for
18: return  $(R, P)$ 

```

---

such a point is considered to be coincident with the half-space, and is collected not only into the resulting polygon  $R$ , but also into the set of intersection points  $P$ . Intersection points are collected so that the intersection algorithm that handles polyhedron/half-space intersection can close the intersection result with a *cap polygon*. Separation of regular and irregular intersections based on signed distances does not help with duplicate intersection points. However, all points are collected following the orientation of the polygon, so the duplicate intersection points, if any, will be consecutive and  $n$  oriented intersection points that contain consecutive duplicates equal up to  $4\epsilon_g$  can be removed with  $\mathcal{O}(n)$  complexity in the best case.

An intersection between a polyhedron and a half-space is handled as the intersection between the collection of polygons and a half-space, with the addition of checking if the result of the intersection between a polygon  $\mathcal{Q}$  and the half-space  $\mathcal{H}$  is co-planar with the half-space, and orienting the cap polygon built by intersection points so that its normal area vector points outside the result polyhedron  $R$ . Algorithms 14 and 15 do not prescribe how the polygon and the polyhedron should be mathematically modeled. For example, a polyhedron can be modeled using indirect addressing (cf. chapter 3), as an unordered set of polygons (*polygon soup*), or as a graph. For the volume fraction transport performed by algorithm 7, geometrical algorithms are based on linear access to sub-elements of the geometrical model: a polyhedron / half-space intersection iterates over polygons and performs the intersection with the half-space, which in turn iterates over polygon edges (point pairs) to do the same. There is therefore some freedom in deciding upon a geometrical model and therefore a software implementation of the geometrical operations must be easily extensible for different geometrical models, which is achieved in this work and described in section 6.6. All geometrical models in this work are modeled as ordered and unordered collections of sub-models. Modeling geometrical concepts as collections of sub-models results in data multiplication, but



ensures a direct sub-element access: no indirect addressing (index or graph-based) is required and no additional information on the sub-model collection needs to be computed.

---

**Algorithm 15** Polyhedron  $\mathcal{S}$  / half-space  $\mathcal{H}$  intersection
 

---

```

1: Initialize intersection result  $R = \emptyset$ .
2: Initialize intersection polygon  $P = \emptyset$ .
3: for Polygon  $Q \in \mathcal{S}$  do
4:    $(R_Q, P_Q) = \text{INTERSECT}(Q, \mathcal{H})$  ▷ Calls algorithm 14.
5:    $Q$  is coplanar
6:   for Point  $\mathbf{p} \in R_Q$  do ▷ Ignore coplanar polygons: add the cap polygon.
7:      $\phi_{\mathbf{p}} = \text{distance}(\mathbf{p}, \mathcal{H})$ 
8:     if  $|\phi_{\mathbf{p}}| > 2|\epsilon_g|$  then
9:        $Q$  is not coplanar
10:      break
11:    end if
12:  end for
13:  if  $Q$  is not coplanar then
14:     $R = R \cup R_Q$ 
15:     $P = P \cup P_Q$ 
16:  end if
17: end for
18: if  $|P| > 2$  then
19:    $P = \text{ORIENT}(P)$ 
20:   if  $|P| > 2$  then
21:     $R = R \cup P$ 
22:    Add the cap polygon  $P$  to the polyhedron.
23:   end if
24: end if
25: return  $(R, P)$ 

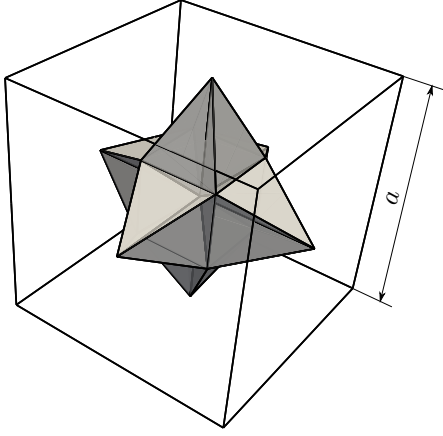
```

---

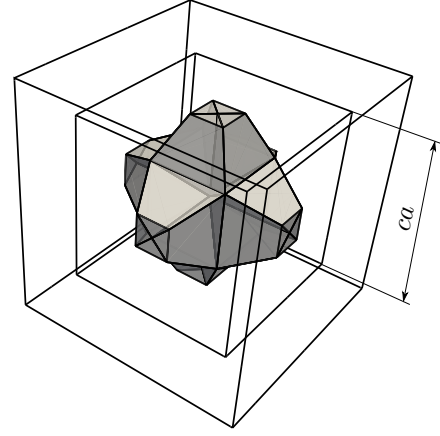
It serves no purpose to outline each and every algorithm from table 2: algorithms 14 and 15 are used as examples in order to describe how the tolerance-based logic is built into the algorithms, that makes sure topologically correct results are generated. Hierarchically higher intersection operations that involve intersections between a polyhedron and a triangulation, a set of triangulations and polyhedra, etc. are simply compositions of algorithms 14 and 15 with the exception of tetrahedral and triangular operations. Tetrahedra and triangles are specialized because both they and their sub-elements have fixed sizes, which can be exploited to increase computational efficiency (e.g. by loop unrolling [84]). Both algorithm categories are equivalent with respect of their respective operations. Where there is no gain in computational efficiency, algorithms that deal with tetrahedra and triangles forward the call to respective algorithms that operate on polyhedra and polygons.

In order to ensure topological correctness, accuracy and robustness, all the algorithms from table 2 were tested against special cases as well as randomly perturbed input data. An example robustness test is shown in figure 37.

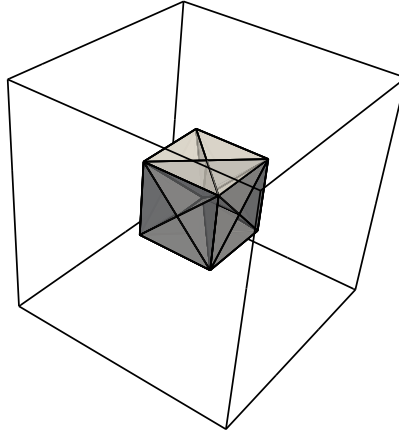
Figure 37 shows an intersection between a star-shaped polyhedron and a cube. The star-shaped polyhedron is built from a cube whose volume is corrected using



(a) Initial star shape / cube intersection.



(b) Star shape / cube intersection.



(c) Co-planar star shape / cube intersection.

Figure 37: Parameterized tests for the perturbed intersection between a triangulated star shape and a cube, where the volume is exactly known.

the pyramidal volume correction shown in section 6.3.3, which makes it possible to know the exact volume of this polyhedron when it is intersected with a cube, because its volume is the sum of the volume of the initial cube and 6 truncated pyramids (c.f. figure 37(b)). The star-shaped polyhedron can be triangulated using the barycentric triangulation because all the triangles of the corrected cap polygons are always visible to the cube centroid. Figure 37(a) shows the initial intersection where the cube is just large enough to touch the points of the star-shaped polyhedron. In this special case, the result should be the entire star-shaped polyhedron itself because it is entirely contained within the cube, even if the position of the cube points is perturbed by  $\epsilon_g$ . The length of the cube edge  $a$  is parameterized by a factor  $c \in [1, \frac{1}{3}]$  and this parameterization is used to iteratively change the cube size, perturb its points, and intersect it with the star polyhedron. The final intersection shown in figure 37(c) is a special case where the faces of the cube are coplanar with the faces of the cube used to build the star shaped

polyhedron. An error in the epsilon-based filtering shown in algorithm 15 could cause the duplication of coplanar polygons, resulting in the error of the volume calculation  $\geq \frac{1}{6}$  of the cube volume. The rest of the 132 geometrical tests involve special case and also perturbed intersections between the geometrical models from the table 2, and they ensure a fast execution speed, robustness and numerical accuracy in the volume calculation down to  $4\epsilon_g$ . For the **UFVFC**, the machine precision was used ( $\epsilon_g = 1e - 15$ ) for geometrical tests.

## 6.5 CONSERVATIVE ERROR REDISTRIBUTION

López et al. [75] were the first ones to observe the large thin flux polyhedra for  $CFL = 1$  in the 2D shear verification case, shown in figure 33(b). The proposed flux-aware triangulation algorithm covered in section 6.3.3 improves the flux volume calculation of the thin flux polyhedra, but there is a limit to its accuracy. Even if the flux polyhedron triangulation triangulates a flux volume without tetrahedral self-intersections, there is no guarantee that this triangulation approximates the flux volume in the best possible way. For example, a negative correction for the volume conservation becomes impossible for the very thin flux polyhedra shown in figure 33(b) without generating self-intersections between tetrahedra, even though all the swept face points may lie on the same side of the face  $\mathbb{F}_f$ . The geometrical calculations used throughout the **UFVFC** scheme rely on a tolerance-based interval logic when dealing with coincidence or collinearity of geometrical models. Additionally, the iterative interface positioning is exact up to a reconstruction tolerance and  $\epsilon_r = 1e - 09$  is used for the results in chapter 11. Consequently, the triangulation of the flux volume, the tolerance-based interval intersection, and the iterative interface positioning are possible sources of errors.

The geometrical **VoF** method attempts to numerically solve the problem of transporting a discontinuity in time and space with second-order convergence and *without any numerical diffusion whatsoever* which is a stiff problem. Assuming that a total error is made by any geometrical **VoF** method in the sum term of equation (179) in the order of magnitude of the reconstruction tolerance  $\epsilon_r = 1e - 09$  on a finest mesh from chapter 11, that has the cell volume magnitudes  $V_c = (\frac{1}{256})^3 \approx 1e - 08$ . The division of the sum term by  $V_c$  amplifies the total error in the phase flux volume calculation by 8 orders of magnitude, resulting in an oscillation in the solution of the magnitude as large as 0.1:

$$\alpha_c^{n+1} = \alpha_c^n - \sum_f \frac{V_f^\alpha}{V_c} + \frac{\epsilon_{V_f^\alpha}}{V_c} \stackrel{V_c \approx 1e-08, \epsilon_{V_f^\alpha} \approx 1e-09}{=} \alpha_c^n - \sum_f \frac{V_f^\alpha}{1e-08} \pm 0.1. \quad (180)$$

The tolerance used by the geometrical operations of the **UFVFC** scheme is set to  $\epsilon_g = 1e - 15$  and it is given by the number of decimal digits available for the double precision by the IEEE 754 double precision standard. If it is assumed that the total phase flux volume calculation error  $\epsilon_{V_f^\alpha}$  is near machine tolerance, the consequence of this assumption is that every single sub-step in algorithm 7 that implements the **UFVFC** scheme is accurate up to machine tolerance. Even then,

equation (180) leads to an oscillation in the solution for  $\alpha_c^{n+1}$  in the magnitude  $\pm 1e - 07$  for every **LE** flux-based geometrical **VoF** method.

The error amplification described by equation (180) can cause the following errors:

- an interface cell is made completely full or empty and vice versa (*wisp*),
- a full cell obtains  $\alpha_c > 1$  (*overshoot*),
- a cell that should have  $\alpha_c < \frac{\epsilon_{Vf}\alpha}{V_c}$  obtains a negative value (*undershoot*).

These errors cause topological changes of the **PLIC** interface: they can create artificial holes and add artificial interface elements. Wisps are especially destructive when they are surrounded by full cells, because in this case cells that are otherwise full are considered as interface cells by the **UFVFC** scheme. If not removed, the wisps propagate upwind relative to the interface and destroy the solution accuracy.

In order to ensure numerical stability and volume conservation, every geometrical **VoF** method relies on one or another form of error correction procedures. It is important to emphasize at this point that there are a few recently proposed geometrical **VoF** methods that do not require any redistribution: like the **CCU** method proposed by Comminal et al. [26] and the method proposed by Owkes and Desjardins [99]. Those geometrical **VoF** methods that are implemented in  $2D$  like the **CCU** and (i)**PAM** can deal more easily with self-intersections and accurate triangulation of envelope flux polygons, than a  $3D$  geometrical **VoF** method like **UFVFC**. Owkes and Desjardins [99] do not mention the need for the redistribution procedure, and it is not clear how the self-intersection of the simplexes caused by correcting the thin flux polyhedra is dealt with, when all the simplexes lie on the same side of a face. Their use of simplex separation for positive and negative flux contributions fixes the self intersection of a flux polyhedron when parts of the flux polyhedron cross the face. The flux polyhedra shown in figure 33(b) and figures 29 and 31 do not cross the face, and yet their volume is wrongly computed using the barycentric triangulation. Consequently, the required volume correction leads to the computation of a larger pyramidal cap, and for thin flux polyhedra, to internal self-intersections of tetrahedra (simplexes) within the flux polyhedron itself. The **UFVFC** scheme still relies on the iterative reconstruction procedure, so the reconstruction error contributes to the error amplification given by equation (180). Therefore, the **UFVFC** scheme still requires a solution stabilization procedure.

As for the other schemes, Jofre et al. [65, algorithm 1] perform volume fraction redistribution to surrounding cells. For the **PAM** and **iPAM**, Zhang and Fogelson [153] points are adjusted and removed in order to ensure numerical stability and volume conservation. Ahn and Shashkov [9, page 2804] rely on both local and global redistribution (repair) on uniform and **AMR** meshes respectively, for their adaptive **MoF** method.

The **UFVFC** scheme is stabilized using a global conservative redistribution algorithm, outlined in algorithm 16. The parallel implementation of the conservative error redistribution algorithm is not as simple as the parallel implementation of the algorithms outlined so far, so this aspect is also addressed by algorithm 16.

The proposed conservative redistribution algorithm computes three types of errors, based on the three aforementioned topological transport errors: wisp volume  $V_\alpha^w$ , undershoot volume  $V_\alpha^-$ , and overshoot volume  $V_\alpha^+$ . Errors in the volume fraction transport are formulated as volumes, as this simplifies volume conservation for the volume fraction redistribution on meshes that have cells with different volume magnitudes. These individual volume errors are then used to compute the total volume error, that is redistributed conservatively to those interface cells that can accept the redistribution volume without artificially creating or destroying interface elements. The conservative redistribution algorithm is separated into three sub-algorithms: process-internal redistribution, process-boundary correction and volume redistribution.

---

**Algorithm 16** Conservative error redistribution

---

- 1: Perform process-internal redistribution.
  - 2: Perform process-boundary correction.
  - 3: Perform volume redistribution.
- 

---

**Algorithm 17** Process-internal redistribution

---

- 1:  $V_\alpha^w = V_\alpha^+ = V_\alpha^- = 0$
  - 2: **for**  $c \in \{1..|\Omega_h|\}$  **do** ▷ Perform process-internal redistribution.
  - 3:    $wisp_c = \text{false}$  ▷ All cells are assumed to be non-wisp cells.
  - 4:   **if**  $\alpha_c < \epsilon_r$  **then** ▷ Remove undershoots and sum  $V_\alpha^-$ .
  - 5:      $V_\alpha^- = V_\alpha^- - (\alpha_c V_c)$
  - 6:      $\alpha_c = 0$
  - 7:   **else**
  - 8:     **if**  $\alpha_c > 1$  **then** ▷ Remove overshoots and sum  $V_\alpha^+$ .
  - 9:        $V_\alpha^+ = V_\alpha^+ + ((\alpha_c - 1)V_c)$
  - 10:        $\alpha_c = 1$
  - 11:     **end if**
  - 12:   **end if**
  - 13:   **if**  $\alpha_c > \left(1 - \frac{\epsilon_{V_f^\alpha}}{V_c}\right)$  and  $\alpha_c < (1 - \epsilon_r)$  **then**
  - 14:      $wisp_c = \text{true}$  ▷ Assume the cell contains a wisp if it is a wisp candidate.
  - 15:     **for**  $d \in N_c$  **do** ▷  $N_c$  : cell-point neighbors.
  - 16:       **if**  $\alpha_d < \epsilon_r$  **then**
  - 17:          $wisp_d = \text{false}$  ▷ Wisps have no empty neighbors.
  - 18:       **end if**
  - 19:     **end for**
  - 20:   **end if**
  - 21: **end for**
- 

Process-internal redistribution algorithm 17 is used for isolating all cells with overshoots and undershoots and correcting their values. All wisp cells that are local to the parallel process are flagged as wisp cells if their value falls within a threshold given by  $\frac{\epsilon_{V_f^\alpha}}{V_c}$ . As outlined in section 6.3.4, the term  $\frac{\epsilon_{V_f^\alpha}}{V_c}$  cannot be

directly computed. Therefore, algorithm 17 relies on a user-specified wisp tolerance

$$\epsilon_w \approx \frac{\epsilon V_f^\alpha}{V_c}. \quad (181)$$

For all the results shown in chapter 11, the wisp tolerance was set to  $\epsilon_w = 0.1$ , based on the observations made in section 6.3.4, because the reconstruction tolerance was used  $\epsilon_r = 1e - 09$ .

Some cells can be falsely categorized as wisp cells if they are local to a parallel process but adjacent to a process boundary and have empty ( $\alpha_c < \epsilon_r$ ) process-neighbors. Algorithm 18 is the process-boundary correction algorithm that is executed after algorithm 17, that isolates and correctly flags wisp cells next to process boundaries.

As for the UFVFC scheme algorithm 7 and the DGNR algorithm 6 used for the interface reconstruction, algorithm 16 relies on the indirect addressing of the unstructured mesh (chapter 3). In order to access cell values from the neighboring process for those cells that have a point in common with the face  $f$ , indirect mesh addressing that is *local to the process* is simply applied on the list of volume fraction values of the cells adjacent to the process boundary  $\alpha_q$ . The process boundary is a set of mutually related faces: the unstructured mesh provides the possibility to obtain a list of all faces that are connected with the face  $f$  via any of its points (face-point faces):

$$N_{f,c} = \{g : \mathbb{F}_g \cap \mathbb{F}_f = \mathbf{p}, \mathbf{p} \in \mathbb{P}\}. \quad (182)$$

Note that  $\forall g \in N_{f,c} \exists \alpha_g \in \alpha_p$ : for each face of the process boundary there exists a volume fraction of the process-local boundary field. The key idea in an efficient parallel implementation of the redistribution algorithm using domain decomposition is that the addressing is exactly the same for the exchanged process-neighbor boundary field  $\alpha_q$ , and can therefore be re-used to inspect the volume fraction values of the process-boundary adjacent cells from the neighboring process.

**Algorithm 18** Process-boundary correction

---

```

1: for  $p \in \{1..|\partial\Omega_h|\}$  do                                 $\triangleright$  Find false wisp cells next to process boundaries.
2:   if  $\partial\Omega_h^p$  is a process boundary then
3:     Exchange volume fraction fields at process boundaries  $(\alpha_p, \alpha_q)$ .
4:     for  $f \in \{1..|\alpha_q|\}$  do                                 $\triangleright \alpha_q$  : process-neighbor volume fractions.
5:       if  $wisp_{owner}(f)$  then                                 $\triangleright$  If the owner cell is a wisp cell.
6:         for  $g \in N_{f,c}$  do                                 $\triangleright N_{f,c}$ : face-point faces,  $\{g : \mathbb{F}_g \cap \mathbb{F}_f = \mathbf{p}, \mathbf{p} \in \mathbb{P}\}$ 
7:           if  $\alpha_{q,g} < \epsilon_r$  then                         $\triangleright$  Wisps have no empty process-neighbors.
8:              $wisp_{owner}(g) = \text{false}$ 
9:             break
10:          end if
11:        end for
12:      end if
13:    end for
14:  end if
15: end for

```

---

Once the falsely marked wisp cells adjacent to process boundaries are corrected by algorithm 18, algorithm 19 performs the actual conservative error redistribution. The global redistribution redistributes the volume error using weighted redistribution of the total volume error defined as

$$V_\alpha^{tot} = V_\alpha^+ - V_\alpha^w - V_\alpha^-. \quad (183)$$

Using the total redistributed volume instead of correcting for individual volume errors has the benefit of reducing the overall error magnitude and improving volume conservation, because it essentially fills the wisp and undershoot cells with the volume available from the overshoot cells. Global redistribution scales better in parallel than the local redistribution, because the local redistribution requires two series of messages to be exchanged: first one to identify the wisp cells and the receiving cells in a wide unstructured cell stencil, and the second one to send the corrected values to the neighbor process. Additionally, a parallel implementation of the local redistribution requires one more step that maps the sent corrected values to the process-local cells. Also, if the redistribution is based on the local cell stencil, it is more likely that the error receiving candidates may not be available in the stencil, in which case either a volume conservation error or numerical boundedness error remains even after the correction procedure. Global redistribution not only guarantees volume conservation up to machine tolerance and exact numerical boundedness, it is also more efficient as it only broadcasts two scalar values to achieve parallel computation: the total volume and the process-global sum of the weights. The global correction also does not rely on frequently executed stencil search loops that redistribute volume errors (Harvie and Fletcher [53, figure 6], Jofre et al. [65, algorithm 1]).

**Algorithm 19** Volume redistribution

---

```

1: for  $c \in \{1..|\Omega_h|\}$  do
2:    $w_c^w = 0$ 
3:   Initialize the wisp weight.
4:   if  $wisp_c = \text{true}$  then
5:      $V_\alpha^w = V_\alpha^w + ((1 - \alpha_c)V_c)$ 
6:   else
7:     if  $wisp_c = \text{false}$  and  $(\alpha_c > \frac{\epsilon V_f^\alpha}{V_c})$  and  $(\alpha_c < (1 - \frac{\epsilon V_f^\alpha}{V_c}))$  then
8:        $w_c^w = 1$  ▷ Redistribution does not create or destroy interface.
9:     end if
10:  end if
11: end for
12:  $w_c^w = \frac{w_c^w}{\sum_{\tilde{c} \in \Omega_h} w_{\tilde{c}}^w}$ ,  $V_\alpha^{tot} = V_\alpha^+ - V_\alpha^w - V_\alpha^-$ 
13: for  $c \in \{1..|\Omega_h|\}$  do ▷ Redistribute total error volume  $V_\alpha^{tot}$ .
14:   if  $wisp_c = \text{false}$  and  $(\alpha_c > \frac{\epsilon V_f^\alpha}{V_c})$  and  $(\alpha_c < (1 - \frac{\epsilon V_f^\alpha}{V_c}))$  then
15:      $\alpha_c = \alpha_c + w_c^w \frac{V_\alpha^{tot}}{V_c}$ 
16:   end if
17: end for

```

---

## 6.6 SOFTWARE DESIGN AND GEOMETRICAL TRANSPORT

This section addresses the software design of the **UFVFC** scheme which was implemented using the C++ programming language. The motivation for this description is twofold: method implementation turned out to be a very challenging task that is worth reporting about, and the software design rationale is required because the source code of the **UFVFC** scheme is prepared to be released to the public domain.

The implementation of the **UFVFC** schemes is based on generic programming using C++ templates. Software design using C++ templates is covered in detail in the following textbooks: Stroustrup [130], Josuttis [67], Vandevoorde and Josuttis [144] and Alexandrescu [12]. An overview of the common techniques used in generic programming can be found in [2]. Background knowledge of generic design using C++ templates is required to understand the software design of the proposed geometrical **VoF** method.

An efficient and still modular and easily extensible implementation of the **UFVFC** using the C++ programming language has proven to be a complex task, because the **UFVFC** scheme relies on different algorithms to be executed on the level of mesh faces and edges. The **CFD** simulations and even the larger verification cases may involve millions of cells per core, resulting in millions of calls to different sub-algorithms of the **UFVFC** scheme. Additionally, keeping the sub-algorithms easily exchangeable is crucial for developing scientific software, for two reasons. First, when a numerical method is developed, there is a very high degree of methodological uncertainty. Second, the modularity and extensibility of the method implementation is a prerequisite for combining the sub-algorithms of the developed numerical method in the process of new method development.



Unfortunately, ensuring fast serial execution and modularity at the same time is difficult if the software design of the **UFVFC** scheme is based on **OOD** using dynamic polymorphism. Dynamic polymorphism in the C++ programming language represents objects as collections of data with *pointers to member functions* (so called *virtual member functions*) that implement the sub-algorithms of the **UFVFC** scheme. Since the type of the object is not known at compile time, resolution of virtual member functions (e.g. selecting a flux polyhedron triangulation) is performed as the program runs. In case of the **UFVFC** the algorithm selection process is executed millions of times per time step for larger problems. Agner [4, section 7.20] states that a virtual function call costs a few CPU cycles more than the call to the virtual function if the function does not change (i.e. the object type does not change) as the program runs. For the **UFVFC** scheme this cost may be significant, because of the very large number of executions involved. Agner [4] proposes the use of static polymorphism based on C++ templates (also called generic programming), instead of dynamic polymorphism in the core parts of the implementation as a solution for the virtual function call overhead.

The call overhead of virtual functions is not the only reason for using generic programming for the implementation of the **UFVFC** scheme. Modern C++ software libraries already exist that perform abstract geometrical computations, like the **BGL** and **CGAL**, and they are both implemented using generic programming. The reasoning behind this design decision is in the general combinatorial flexibility that generic programming supports when it comes to exchanging sub-algorithms as well as models and data structures used by the algorithms. Consequently, generic programming was used for the implementation of the **UFVFC** scheme and its supporting geometrical and transport software framework. The **UFVFC** scheme models a generic concept of a *geometrical transport* method. Therefore, the implementation of the **UFVFC** is split into two generic libraries: the generic geometrical library and the generic transport library.

### 6.6.1 A C++ generic geometrical library

As already noted, the **BGL** and **CGAL** library both rely on generic programming. However, to support the **UFVFC**, a new standalone generic library for 3D geometrical computations was implemented, specialized for 3D triangulations and intersections required by the geometrical **VoF** method.

**CGAL** models non-convex polyhedra and their boolean operations with the Nef-polyhedron model, as described by Hachenberger et al. [51]. Timings of the numerical experiments performed by Hachenberger et al. [51] report run times with the order of magnitude of 10 seconds for the order of magnitude 1000 of vertices in the result of a boolean operation. Therefore, the use of the 3D boolean intersections **CGAL** is prohibitively expensive for the **UFVFC** scheme. The benefit of Nef polyhedra lies in handling topological special cases that may arise from boolean operations, which has its purpose in some applications. For the geometrical **VoF** method, however, much less generality and much higher execution speed is expected from the supporting geometrical operations. The **BGL** is dimension

agnostic, but most of the geometrical operations of the **BGL** are implemented in *2D*. Furthermore, **BGL** relies on type-based tag dispatching to resolve ambiguities in function signatures for different sets of template arguments [48]. Type-based tag dispatching relies on partial template specialization compared to overloading which is used for resolving function signatures by value-based tag dispatching. Using partial template specialization for tags reduces the amount of function parameters required to resolve the function signature. However, partial template specialization requires the use of structure or class templates, because partial function template specialization is not allowed by the current standard of the C++ language. When tag dispatching is used, only the tag type template arguments must be resolved, the rest of the template parameter remains under-resolved, so that it can be used for any template parameter set that correspond to the resolved tag argument set. Therefore, every algorithm category in **BGL** requires a general function template to be implemented, that then dispatches a call to the dispatch namespace, where static member function of class (structure) templates are placed, that contain the actual implementation.

Alternative to the design used for **CGAL** and **BGL**, Substitution Failure Is Not An Error (**SFINAE**) together with template metaprogramming can be used as the means for resolving function signatures, which can be an elegant solution when the template metaprogramming functions available in the C++11 standard are used, and this solution was adopted for the **UFVFC** scheme. Some preliminary discussion on different design options is required in order to support the decision of using **SFINAE** and metaprogramming, since both techniques are often avoided by programmers because of their perceived complexity.

High level of abstraction offered by generic programming as well as the freedom in easily combining data structures and related algorithms provides the foundation for fast optimization for computational efficiency as well as the possibility to exchange geometrical models in order to improve solution correctness. One of the main techniques of generic programming known as *type lifting* could be used to achieve this goal. Type lifting is based on the fact that the generic code prescribes requirements on the types involved in the calculations (template arguments). Examples of such requirements on template arguments are: copy-constructible, assignable, destructible, provides direct access to its elements, provides public access to specific attributes, implements iterator interface, etc. Requirements enforced on template arguments by the generic code define a single implicit interface for each argument.

The implicit interface defines an abstract notion - a concept: a set of requirements for the template parameter. A concept is not implemented in terms of source code and is therefore fully abstract. An *example of a concept* would be a polygon sequence: a sequence of polygons that provides direct access to each polygon and implement an Standard Template Library (**STL**) iterator interface. An *example model* of the polygon sequence concept would be an **STL** vector of polygon models.

Template arguments either satisfy the implicit interface prescribed by the template, or they do not. Those template arguments that satisfy the implicit interface can be used to instantiate a template, resulting in valid code. When an error occurs during the substitution of the template argument, the compiler *does not produce an*

*error and stop the compilation*, it continues to search through remainder of the set of available templates until such template is found, for which the argument satisfies the implicit interface. This standard behavior is known as Substitution Failure Is Not An Error (**SFINAE**) (Stroustrup [130, section 23.5.3.2]) and the implementation of the proposed geometrical **VoF** method relies on it.

A trivial example of the ambiguity faced by the compiler when only type lifting is applied to the geometrical algorithm is shown in source code listing 6.1. Merely naming the template parameters differently does not make them different to the compiler. In the code shown in source code listing 6.1, the algorithm does not actually compute the volume of a polyhedron. The computation is valid *for any model* that implements an **STL** iterator interface and for which the `volume(C, p)` algorithm is implemented. Defining multiple function templates such as one in source code listing 6.1 results in an ambiguity error.

Listing 6.1: Ambiguous volume calculation

```
template <typename Polyhedron, typename RT = double>
RT volume(Polyhedron const & p)
{
    RT result(0);

    auto C = centroid(p);

    for (const auto& polygon : p)
        result += volume(C, p);

    return result;
}
```

In order to disambiguate algorithms by properly resolving function signatures, the following approaches can be applied: tag dispatching and concept-based function overloading. In order to justify the decision for using the concept-based function overloading approach in the implementation of the proposed geometrical **VoF** method, both approaches are outlined in the following sections.

#### 6.6.1.1 Tag dispatching

Selection of algorithms can be enforced based on the specific properties of a template argument, so-called *type traits*. Multiple traits can be related to each other using inheritance, allowing easy combination of their associated properties. Algorithms that are selected based on type traits might be those that are optimized for efficiency (e.g. specialized for a data structure or a computer architecture), or those that result in better accuracy (e.g. specialized for more accurate floating point representation or a more accurate mathematical model). An example application of trait classes is the selection of an optimal distance calculation between two iterators in the **STL**, described by Josuttis [67, section 9.3.3, page 445].

An example trait class is shown in source code listing 6.2, together with two concrete tag types: `polygon_collection_tag` and `polygon_sequence_tag`. Tag dispatching relies on a general function template, that obtains the tag of the template arguments from the tag trait class shown in source code listing 6.2 and

Listing 6.2: Tag trait structure.

```

struct polygon_collection_tag {};
struct polygon_sequence_tag : polygon_collection_tag {};

// A general tag is not allowed.
template<typename Geometry>
struct tag;

// A specialization of a polygon sequence tag.
template<>
struct tag<PolygonSequence>
{
    typedef polygon_sequence_tag type;
};

```

Listing 6.3: value-based tag dispatch.

```

template
<
    typename GeometryOne,
    typename GeometryTwo,
    typename RT = double
>
RT volume(GeometryOne const & one, GeometryTwo const & two)
{
    return volume(
        one, // Forward first geometry.
        tag<GeometryOne>::type(), // Forward its tag by value.
        two, // Forward second geometry.
        tag<GeometryTwo>::type() // Forward its tag by value.
    );
}

```

then dispatches the function call further. The way the function call is dispatched is determined by the choice of the tag dispatching approach: value-based or type-based dispatching.

Value-based tag dispatching is shown in source code listing 6.3 and relies on function overloading for tag objects. The function overloading allows the tag dispatching system to introduce co-dependence between tags, such as inheritance which is shown in source code listing 6.2. For example, a polygon sequence is a collection of polygons, and this relationship is modeled with the inheritance relationship between the respective tags. When value-based dispatching is applied, as in the return statement in source code listing 6.3, that volume algorithm will be chosen among all candidates for which the tag objects are most specific. The derived tag types are given precedence by the overloading process, as they are more specific.

Type-based tag dispatching uses tag types as template arguments for class (or structure) templates and relies on the partial template specialization to provide alternative implementations. The type-based tag dispatching approach is used by the boost.geometry library, and is covered in detail by Gehrels and Lalande [46] and Gehrels et al. [47], as well as the official library documentation [1].

When type-based tag dispatching is used, structure templates are implemented within a separate (dispatch) namespace and calls to those algorithms are made

Listing 6.4: Type-based tag dispatch.

```

template
<
    typename GeometryOne ,
    typename GeometryTwo ,
    typename RT = double
>
RT volume(GeometryOne const & one , GeometryTwo const & two)
{
    return dispatch :: volume
    <
        typename tag<GeometryOne > :: type ,
        typename tag<GeometryTwo > :: type
    > :: apply(one , two);
}

```

by the generic dispatch function template similar to one in source code listing 6.3. source code listing 6.4 holds an example of a type-based dispatch.

The tag dispatching shown in source code listing 6.3 and source code listing 6.4 are similar in the fact that they rely on tags to disambiguate the algorithms that perform the calculations. The difference lies in the fact that the algorithms are implemented as static member functions for type-based tag dispatching, and global functions for value-based tag dispatching. Extending the algorithm set with a global function incurs less programming overhead than extending the dispatch namespace with a specialized structure template. Keeping the programming overhead low for structural enhancements of the implementation plays an important role both in maintenance and future extensions of the geometrical Volume-of-Fluid method.

One issue with value-based tag dispatching is that the number of function arguments grows with the introduction of tags. On the other hand, with type-based tag dispatching, extending the set of available algorithms requires the user to implement a structure within the dispatch namespace. Both tag dispatching approaches also complicate the code for those algorithms that call on other algorithms. At each call site (e.g. polygon intersection called from within polyhedron intersection), a tag dispatch call must be implemented which results in surplus code that is used only to disambiguate algorithms.

#### 6.6.1.2 *Concept-based function overloading*

The concept-based overloading is not to be misinterpreted with concepts proposed by B. Stroustrup and A. Sutton et al. [16]. At the point of writing this text, C++ concepts represent an experimental language extension which is not yet supported by the language standard. Therefore, it has been decided not to disambiguate geometrical algorithms using C++ concepts. However, the proposed design of the geometrical library would require small changes in order to introduce C++ concepts. This claim is based on the fact that the metafunctions involved in the compile-time return type calculations are placed in the function return type deduction line, and can therefore be easily exchanged with concepts if they are accepted as the part of the C++ standard.

Listing 6.5: Polygon sequence / half-space distance.

```

template
<
    typename PointSequence ,
    typename Halfspace ,
    typename RT = double
>
typename return_type // SFINAE return type resolution.
<
    Polygon ,
    point_sequence_tag ,
    Halfspace
    halfspace_tag
>::RT
distance(Polygon const & poly , Halfspace const & hspace)
{
    RT result ;

    // Operations on poly and hspace.

    return result ;
}

```

Listing 6.6: Indexed polygon / half-space distance.

```

template
<
    typename IndexedPolygon ,
    typename Halfspace ,
    typename RT = double
>
typename return_type<IndexedPolygon , Halfspace >::RT
distance(IndexedPolygon const & poly , Halfspace const & hspace)
{
    RT result ;

    // Operations on poly and hspace.

    return result ;
}

```

Concept-based overloading in the form of function overloading based on arbitrary properties of types as introduced by Järvi et al. [60] has been considered as a tool to disambiguate geometrical algorithms. The basic idea behind this mechanism is to use template metaprogramming to resolve return types for the function templates at compile time. In cases when the metafunction fails to compute the function return type, the **SFINAE** idiom forces the compiler to check other available function templates for a viable choice.

Source code listings 6.5 and 6.6 show two different template functions that implement the distance calculation between the polygon and half-space concepts. The example metafunction **return\_type** returns either a specific return type-based on properties of the parsed template arguments, or a void type. Since the functions return a result, a void type cannot be used and the template argument substitution fails to produce valid code.

The metafunctions responsible for calculating the return type can be based on the implicit interface defined by the function implementation. Without the

availability of C++ concepts at this point, this would involve writing metafunctions that check the interface of the template argument. The problem is that in that case the function resolution ambiguity is still not resolved in cases when multiple algorithms exist that prescribe the same implicit interfaces, as discussed for the source code listing 6.1.

The way the metafunction `return_type` is implemented determines the behavior of the template instantiation process. Järvi et al. [60] have shown that the metafunctions can be combined with tags. In that case, no general function template is required that implements tag dispatching, as shown in source code listings 6.3 and 6.4 for value-and-type-based tag dispatching, respectively. However, the benefit does not lie in removing a single generic function template for tag dispatching per set of algorithms. The main benefit lies in the dispatch call being completely removed from sub-algorithm call sites. This significantly increases code readability, reduces code bloat and improves software design scaling.

Listing 6.7: Polygon sequence / half-space intersection

```

template
<
    typename IntersectionResult = polyhedronIntersection ,
    typename IntersectionTest = no_test ,
    typename PolygonIntersection = polygonIntersection ,
    typename PolygonSequence ,
    typename Halfspace
>
typename std::enable_if
<
    tag_enabled
    <
        PolygonSequence ,
        polygon_sequence_tag
    >::value &&
    tag_enabled
    <
        Halfspace ,
        halfspace_tag
    >::value ,
    IntersectionResult
>::type
intersect(
    PolygonSequence const & polyhedron ,
    Halfspace const & halfspace
)
{
    ...

    for (const auto& polygon : polyhedron)
    {
        // No tag dispatching required for sub-algorithms.
        auto intersection = intersect<PolygonIntersection>(
            polygon ,
            halfspace
        );

        ...
    }
}

```

Software design scaling can be defined as the number of points in the source code that need to be modified (points of variation) as the function of the number of required extensions. In cases when the new geometrical model can be efficiently used with existing algorithm function templates, there is one point of variation in the



concept-based generic geometrical library: tagging the model with the appropriate tag. When a new algorithm is to be added that works with existing concepts, there is one point of variation: implementing a function template, enabled by the existing tags. In the case when both a new model and a new algorithm are to be added, there are two points of variation: introducing a new tag, and programming a new function template. It can therefore be concluded that the applied design for the geometrical library scales very well.

Very good scaling is the reason behind using the concept-based design proposed by Järvi et al. [60] for the implementation of the geometrical library. Each concept involved in the geometrical calculation is tagged with a tag. An example of a concept is a polyhedron described as a sequence of polygons. A sequence has restrictions regarding the data structure used to store the polygons - it needs to be compliant with an `STL` sequence container. A part of an intersection algorithm based on such a concept is shown in source code listing 6.7. The intersection algorithm has been made generic with respect to the intersection result, test for intersection, as well as the polygon sequence and half-space geometrical concepts. The metafunction is responsible for checking if the arguments have been tagged with expected tags. Note the call to the polygon intersection sub-algorithm in the final line of the listing. The call site is expressive and short, since no tag dispatching is required. The metafunction is shown in full detail only for better clarity - the actual metafunction implementation can be made significantly shorter.

There are many benefits of using function overloading based on argument properties combined with type tags for the geometrical library. Any geometrical algorithm, e.g. the source code listing 6.7, can be re-used with any model that is tagged as a polygon sequence. Multiple algorithms are disambiguated easily without adding dispatch namespaces and structure templates. The design scales well - there is only one point of variation per algorithm and a new concept, respectively. Different data structures and sub-algorithms can be interchanged without the overhead and complexity brought by inheritance and dynamic polymorphism: changing the `PolygonIntersection` type in source code listing 6.7 requires simply an invocation of `intersect` with a different template argument.

#### 6.6.1.3 *Object oriented design as an alternative*

Even though generic design is applied for the geometrical Volume-of-Fluid method, it is worthwhile to comment on the object oriented design as an alternative. The geometrical algorithms are called by the transport algorithms millions of times, even for simulation cases with the number of cells  $\approx 1e05$ . Geometrical operations that are executed most often are: the tetrahedral decomposition (triangulation), the polyhedron/half-space intersection and the flux polyhedron correction (volume conservation correction). As already noted, the operations have very short execution times (atomic operations), imposing dynamic polymorphism using object oriented principles incurs a noticeable overhead in the overall computational time of the transport algorithm.

Dynamic polymorphism imposes a function call overhead, since the code of the function to be executed is to be chosen at execution time, using virtual pointer tables



(vtables). The vtables are stored by any object that uses dynamic polymorphism and they point to the code of the called function. A detailed overview of the relative performance cost of the polymorphic routine call to other common operations in the C++ language is given by McConnell [84], on page 601. Additional to the function call overhead, virtual functions are mostly not inlined by the compiler as it performs efficiency optimizations. Implementing the algorithms as function templates enables the compiler to perform extensive efficiency optimizations.

The object oriented design additionally complicates the process of combining different strategies within the geometrical algorithms. As described by Alexandrescu [12], in cases when the combinations are to be addressed with multiple inheritance, the design does not scale well. The issue of combining policies is handled automatically with a generic design simply by using different template arguments.

One could argue that the generic design takes away the possibility to provide the selection of algorithms and models at run-time which results in a lower flexibility in use. The implementation of the geometrical Volume-of-Fluid method prevents that on purpose for the low-level geometrical calculations because of the aforementioned reasons. As for the transport algorithms and the corresponding models, an object oriented extension is very straightforward. Once the most efficient combination of algorithms and data structures of the geometrical method has been selected, the elements of the method can be easily made run-time selectable. Instantiated templates are then wrapped using the Adaptor Pattern [44], where the adaptor provides run-time selection capability.

Applying OOD for a selected subset of transport algorithms and the corresponding models represents no issue when it comes to computational costs. The transport algorithms operate on the level of the CFD solution algorithms and require orders of magnitude more time to complete, than the low-level geometrical algorithms. Therefore, enabling dynamic polymorphism for a selected sub-set of transport algorithms results in a greater flexibility in use and does not incur a visible computational overhead. In other words, the function call overhead in selecting at run-time an adapted instantiation of the reconstruction algorithm template code is negligible compared to the time it takes to reconstruct the geometrical interface with hundreds of thousands elements.

### 6.6.2 A C++ generic transport library

The generic geometrical transport library abstracts the algorithms and models related to the geometrical transport of the volume fraction field  $\alpha_c$ . Introducing concepts by overloading functions based on arbitrary properties (tags, traits) assigned to types (template arguments) used for the geometrical calculations and covered in section 6.6.1 is not necessary for the geometrical transport, because a single concept emerges within the geometrical VoF method: the PLIC interface. As the volume fraction advection algorithm 7 requires random access to interface elements, the interface elements must provide this access with  $\mathcal{O}(1)$  complexity. This does not leave the design with much freedom in representing the PLIC interface

using different data structures, as is the case for the generic geometrical library, whose algorithms are linear with respect to the model elements and do not require direct element access. Consequently, there is no need to use concept-based design of Järvi et al. [60] for the geometrical transport.

Listing 6.8: Swartz interface reconstruction

```
template
<
  typename TransportControl ,
  typename InterfacePositioning ,
  typename InterfaceSizeAdjustment
>
class SwartzReconstruction
:
  public YoungsReconstruction
  <
    TransportControl ,
    InterfacePositioning ,
    InterfaceSizeAdjustment
  >
{
  ...
}
```

A generic implementation of a geometrical **VoF** method that combines the algorithms outlined in sections 6.2 and 6.3 can be based on the much simpler, *policy-based design* Alexandrescu [12, chapter 1]. Source code listing 6.8 shows the implementation of the Swartz reconstruction algorithm described in section 6.2. Sub-steps of the algorithm that can be completely generalized are abstracted into policies: algorithm control that stores important tolerances, interface positioning, and the interface size adjustment policy for efficient storage re-allocation. As described by Alexandrescu [12, chapter 1], the policies implement sub-algorithms used by the host class template: in this case the reconstruction algorithm class template.

Listing 6.9: **UFVFC** advection

```
template
<
  typename TransportControl ,
  typename FluxPolyhedronGenerator ,
  typename FluxStencilGenerator ,
  typename PhaseFluxVolumeCalculator
>
class UnsplitAdvection
:
  public FluxPolyhedronGenerator ,
  public FluxStencilGenerator ,
  public PhaseFluxVolumeCalculator
{
  ...
}
```

, The **UFVFC** scheme is implemented by the **UnsplitAdvection** class template outlined in source code listing 6.9, with policies for all the sub-algorithms of algorithm 7: transport control, flux polyhedron generation with volume correction, flux stencil generation and phase flux volume calculation. Policy-based design is applied

for the policies themselves as well. The transport policies use the generic geometrical library for all the geometrical operations required by the geometrical **VoF** method. Source code listing 6.10 contains a few lines of code of the phase volume calculation algorithm. Algorithms (`build`, `triangulate`, `barycentric_triangulate`, `do_intersect`) and models (`halfspaceVector`, `triangleVector`) from the geometrical library are used easily. At all call points where the return type is the default type declared for the policy algorithm (e.g. `triangulate`, `build`, etc), the return type is not explicitly specified.

Listing 6.10: Phase flux volume calculation

```
// Triangulate the flux polyhedron.
auto fluxPolyhedronTriangulation = triangulate(fluxPolyhedron);
// Triangulate the swept face points.
auto displacedTriangles = barycentric_triangulate(displacedPoints);

// Compute the phase flux volume contributions.
for (const auto cellLabel : cellLabels)
{
    if (control_.isPhaseValue(volFractionField[cellLabel]))
    {
        // Reformulate a cell as a set of halfspaces.
        auto cellHalfspaces = build(cellLabel, mesh);

        std::vector<int> cuttingHalfspaces;
        cuttingHalfspaces.reserve(cellHalfspaces.size());

        // Select a set of halfspaces that intersect the displaced points.
        for (decltype(cellHalfspaces.size()) hI = 0;
             hI < cellHalfspaces.size(); ++hI)
        {
            for (const auto& triangle : displacedTriangles)
            if (do_intersect(triangle, cellHalfspaces[hI]))
            {
                cuttingHalfspaces.push_back(hI);
                ...
            }
        }
    }
}
```

### 6.6.3 Conclusions

Using the design of function overloading with arbitrary properties of types proposed by Järvi et al. [60] resulted in an implementation of a completely generic geometrical library that can be easily extended with new concepts and algorithms with a low programming overhead. The generic geometrical library implements the algorithms and models outlined in table 2. Algorithms that implement functions of two variables are implemented for the pairs of models that are required by the **UFVFC** scheme. Very good scaling of the design proposed by Järvi et al. [60] supported by simplified template metaprogramming available in the C++17 standard.

Using the policy-based design has lead to the possibility to easily exchange individual policies of the dimensional unsplit volume fraction advection scheme. In fact, the **UFVFC** scheme is simply an instantiation of the **UnsplitAdvection** class template with a specific set of policies. In order to implement an alternative scheme to the **UFVFC**, a new **FluxVolumeCalculator** policy needs to be implemented and the template argument accordingly renamed at the instantiation point. Using this design, new methods for the geometrical transport can be assembled from the

existing set of policies without any modifications of the existing code. This was ensured by selecting those sub-algorithms of algorithm 7 for policies, that are independent of each other (i.e. *orthogonal policies*, so named by Alexandrescu [12]).

In conclusion, in order to implement the **UFVFC** scheme, two efficient and yet modular software layers (geometrical and transport) were implemented with software design practices that support straightforward extensions both in terms of new functionality, as well as combining existing algorithms, without incurring large programming or computational overheads.

## 6.7 A NOTE ON THE CFL CONDITION

The **UFVFC** scheme takes on an explicit, multidimensional and geometrical approach to approximating the solution of the volume fraction equation (59). Because of its explicit nature, the scheme is subject to the **CFL** criterion [55, 91]. Since the **UFVFC** scheme relies on computing the phase flux volume associated to the face of a mesh cell, the **CFL** criterion is computed as a face-centered value using

$$CFL_f = \frac{F_f \Delta t}{\|\mathbf{x}_{P,f} - \mathbf{x}_{N,f}\| \|\mathbf{S}_f\|}, \quad (184)$$

where  $CFL_f$  is computed using the face centered velocity magnitude approximated from the volumetric flux

$$\|\mathbf{U}_f\| \approx \frac{F_f}{\|\mathbf{S}_f\|}. \quad (185)$$

The maximal face-centered **CFL** number in the whole domain is used to restrict the global time step of the simulation

$$CFL = \max(CFL_f). \quad (186)$$

A purely geometrical **CFL** criterion has not been considered in this work, because this would incur additional search operations on unstructured meshes to ensure that the flux polyhedron does not exit the geometrical bounds of the flux stencil  $N_{F_f}$ .

## **Part III**

### **Hybrid Level Set / Front Tracking Method**



---

METHOD OVERVIEW

---

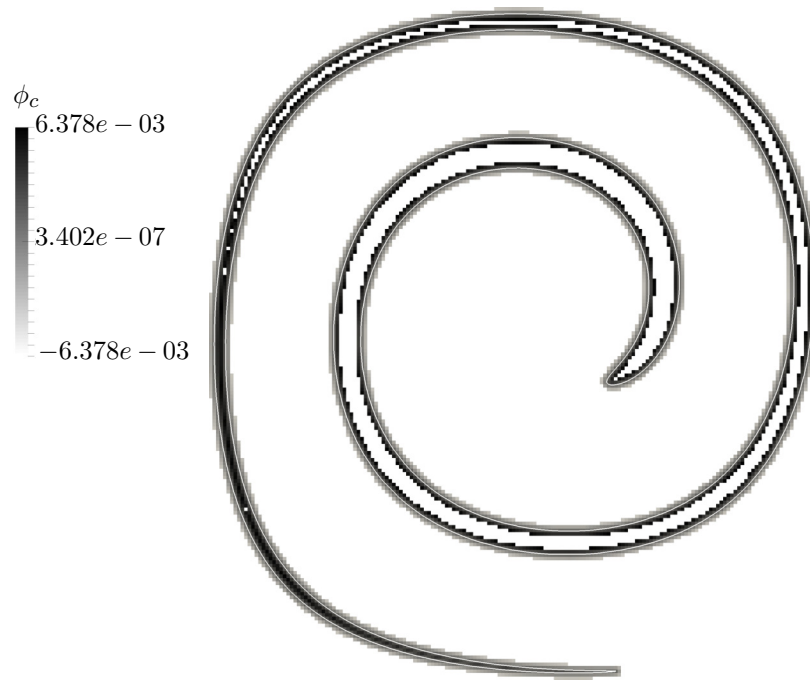


Figure 38: **LENT** signed distance  $\phi_c$  calculated in the narrow band for the 2D shear verification case with  $256^2$  cells,  $CFL = 1$ . Interface (surface mesh) is colored white.

Recent publications show a trend towards developing so-called *hybrid* methods: combinations of algorithms that alleviate an inherent deficiency in one method by replacing a problematic sub-algorithm with a better algorithm from another method. Some geometrical **VoF** method methods reviewed in section 5.2 confirm this trend. Le Chenadec and Pitsch [71] propose a Hybrid Lagrangian–Eulerian Method for Multiphase flow (**HyLEM**) method that relies on the signed distance field for the calculation of the curvature - a hybrid Level Set / **VoF** method. The **PAM** method proposed by Zhang and Fogelson [153] can be considered as a hybrid Front Tracking / Volume-of-Fluid method, since additional points are distributed along the linear interface that are then tracked in a Lagrangian way in order to ensure fourth-order accuracy. Other contemporary hybrid Level Set / Front Tracking methods are discussed in chapter 8.

The **SMCI** volume fraction initialization algorithm proposed for the geometrical **VoF** method described in chapter 6 intersects the exact interface modeled as a

surface mesh  $\Gamma_h$  with the volume mesh used to discretize the solution domain  $\Omega_h$ . In order to accomplish this task, a signed distance field is constructed within the layer of cells that surround the interface and the inside/outside categorization of mesh points with respect to  $\Gamma_h$  is performed. These steps build the foundation for the developed hybrid Level Set / Front Tracking method.

In order to make it possible to develop hybrid methods in the future and use the advantages the Front Tracking method has for specific physical problems (e.g transport of surfactants on the fluid interface developed by Muradoglu and Tryggvason [92]), a hybrid Level Set / Front Tracking (**LENT**) method is developed on the basis of the operations provided by the **SMCI** algorithm. Alternatively to Front Tracking, the Interface Tracking method developed by Muzafferija and Peric [93] and Tuković and Jasak [140] and extended for **DNS** of the droplet formation process with soluble surfactants by Dieter-Kissling et al. [33] explicitly tracks the interface between two immiscible fluids and allows for a straightforward simulation of transport processes on the fluid interface. The main difference between Front Tracking and Interface Tracking lies in the fact that the Interface Tracking aligns a mesh boundary with the fluid interface, so that the evolution of the interface is described by the evolution of the mesh boundary. As a consequence of this alignment, Interface Tracking has the formal order of accuracy of the underlying numerical method used to obtain the mesh point velocities and the displacements required for the mesh motion - excluding other complexities that may arise in the pressure-velocity coupling algorithm. Front Tracking methods handle the fluid interface as a separate immersed mesh of dimension  $n - 1$ , where  $n$  is the dimension of the solution domain  $\Omega_h$ . The advantage of Interface Tracking is therefore given by the alignment-driven higher accuracy, and the advantage of the Front Tracking method is the ability to impose topological changes on the interface, disconnected from the solution domain  $\Omega_h$ .

The hybrid Level Set / Front Tracking (**LENT**) method models the interface between two fluids using a surface mesh in  $3D$  or a set of mutually connected line segments in  $2D$ . The surface mesh is tracked forward in time along discrete Lagrangian trajectories, and the categorization of intersected/inside/outside cells of the encompassing volume mesh  $\Omega_h$  is used to reconstruct the marker field  $\alpha_c$ . The signed distance field computed in the near vicinity of the surface mesh can be used at any point to reconstruct the interface using an iso-surface reconstruction algorithm. This makes it possible for the **LENT** method to retain control over when and how the topological changes of the interface (coalescence and breakup) are handled. Because the motion of the interface is modeled by evolving the interface mesh in a Lagrangian way that depends on the fields that map to the volume mesh  $\Omega_h$ , hybrid Level Set / Front Tracking methods belong to the Lagrangian tracing / Eulerian remapping (**LE**) method category.

Figure 38 shows the signed distance field surrounding the interface computed by the **SMCI** algorithm. This, however, is not sufficient for two-phase flow simulations, as a cell-centered marker field is required by the single-field two-phase **NS** equation system in order to distinguish between phases in every cell  $\mathbf{C}_c \in \Omega_h$ . Different models exist for reconstructing the marker field from the signed distance field,



however every model requires the signed distance value to be correctly assigned to each cell center.

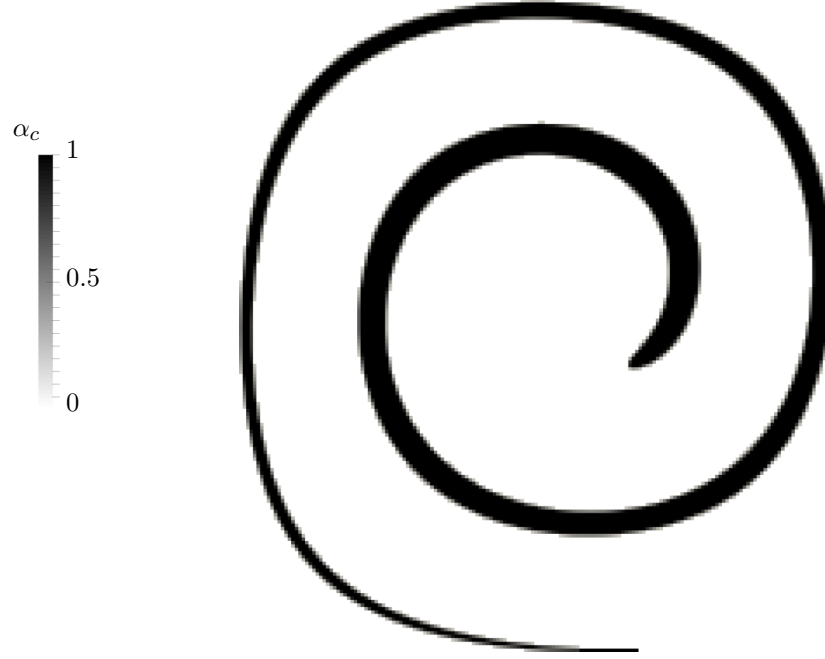


Figure 39: **LENT** marker (volume fraction) field  $\alpha_c$  for the 2D shear case with  $256^2$  cells,  $CFL = 1$ .

From a signed distance field defined in each cell, a marker field is computed, as the one shown in figure 39, and used by the single field two-phase **NS** equation system to compute the velocity. Motion of the interface in the **LENT** method and other Front Tracking methods is based on interpolating the velocities from cell centers to the vertices of the surface mesh. Those interpolations require the mapping  $v \mapsto c$ : each  $v$ -th vertex of the surface mesh must be associated with a cell with index  $c$ . Calculating and maintaining this association for a moving surface mesh immersed in a volume mesh represents the major source of complexity of any Front Tracking method on unstructured meshes and the core contribution of the proposed Level Set / Front Tracking (**LENT**) method.

The following chapters cover the details of the **LENT** method that are only outlined in this chapter. Chapter 8 lists the state of the art hybrid Level Set / Front Tracking methods already reported in the scientific literature and relates the proposed **LENT** method to recent research. Chapter 9 covers the details of all the algorithms used by the **LENT** method on unstructured meshes.



---

LITERATURE SURVEY

---

The Level Set / Front Tracking (**LENT**) method extends the Front Tracking family of methods to unstructured meshes. The Front Tracking method (Unverdi and Tryggvason [143], Jayaraman et al. [63], Glimm et al. [49] and Tryggvason et al. [138]) provides very accurate results for both segregated and dispersed two-phase flows. Additionally, the method has been used to simulate flows involving transport phenomena taking place on the fluid interface. Zhang et al. [150] and Muradoglu and Tryggvason [92] have used the front tracking method to simulate flows with soluble surfactants on the interface. Hua et al. [58] and Roghair et al. [114], among others, have used the Front Tracking method to investigate bubbly flows.

The most prominent alternatives to the Front Tracking method for **DNS** of two-phase flows are the geometrical **VoF** method method (cf. chapter 5) and the Level Set method (Sussman et al. [132, 133], Sethian [122], Osher and Fedkiw [97]). Both methods have specific advantages and disadvantages with respect to each other. Recently, efforts have been made to compare different methods and combine their algorithms in such ways that hybrid methods are constructed which outperform the original methods in both overall accuracy and efficiency. Du et al. [36] show that their grid-based Front Tracking method implementation results with volume conservation errors comparable to those of the Volume of Fluid (VoF) method. Sussman and Puckett [131] successfully couple the Level Set and the VoF method in order to deal with the disadvantage of the VoF method when it comes to accurate surface tension calculation and the problems with volume conservation inherent to the Level Set method. Jemison et al. [64] have coupled the Moment of Fluid and Level Set method adapted for a collocated flow solution, supporting block structured **AMR**.

The Front Tracking method represents the fluid interface as a topologically connected set of interface elements. The front mesh, or so-called *front*, consists of mutually connected lines in 2D and triangles in 3D. In order for the method to retain generality in simulating two-phase flows, it needs to account for possible topological changes (breakup and/or coalescence) of the interface, each time the element positions are updated. A very detailed description of the Front Tracking method is provided by Tryggvason et al. [139]. The mutual connectivity between the front elements is used to deal with topological changes within the framework of the Front Tracking method, by executing complex explicit topological operations on the front. Furthermore, these operations are global with respect to the front, since coalescence or breakup may involve interaction between arbitrary subsets of front elements. Global and explicit topological operations on the front representing

the fluid interface complicate the implementation of the method, especially its parallelization, which is a crucial point as the DNS simulations of two-phase flows often rely on parallel program execution.

In order to deal with the problem of complex global topological operations on the front, Shin and Juric [125] and Cenicerros et al. [22] have combined an iso-contour (iso-surface) reconstruction with Front Tracking. In their implementations (Level Contour Reconstruction Method (**LCRM**) by Shin and Juric [125] and hybrid level set / front tracking (**LEFT**) by Cenicerros et al. [22]), the authors rely on reconstructing the front from a signed distance field using an iso-surface reconstruction algorithm. The rest of the algorithms are kept similar to those of the classical Front Tracking method in case of the LCRM method, and to the level set method in case of the LEFT method. The iso-surface reconstruction enables this hybrid approach to automatically deal with topological changes of the interface, without performing any direct topological operations on the front. Furthermore, as the reconstruction of the front by the iso-surface algorithm is rendered local (compact on domain boundaries), the parallelization of the method can be achieved using a straightforward domain decomposition approach without resorting to a specialized slave-master process communication pattern characteristic for Front Tracking.

In an extended version of their algorithm, Shin and Juric [126] improve the mass conservation on coarse meshes by using a B-spline reconstruction of the interface from the signed distance field. Further extension of the method was done by Shin and Juric [127] by resorting to local grid based Front Tracking (Du et al. [36]) to further improve mass conservation. All aforementioned results show that the volume (mass) conservation errors converge with increased mesh resolution, as the Front Tracking methods are not inherently volume conservative because the motion of the interface is Lagrangian. Cenicerros et al. [22] added block-adaptive mesh refinement (block **AMR**) to the region near the interface in order to increase the overall accuracy and reduce errors in mass conservation.

Other approaches involving hybrid Level Set/Front Tracking algorithms involve the use of the level set function to impose refinement and motion operations onto a mesh within the ALE framework. Along these lines Nochetto and Walker [94] and later Basting and Weismann [17] have used the iso-surface (zero contour) in order to enforce the alignment of finite volume faces to the iso-contour normals using a global optimization algorithm. The algorithm of Nochetto and Walker [94] additionally allows for automatic topological changes of triangular meshes, however the extension to three dimensions is reported to be not straightforward by the authors. Both algorithms do not employ a detached front mesh of a lower dimension than the background Eulerian mesh. Therefore they are not hybrids of the Level Set method developed for two-phase flows by Sussman et al. [134] and the Front Tracking method proposed by Unverdi and Tryggvason [143] for DNS of two-phase flows.

The present contribution extends the front tracking family of algorithms for DNS of two-phase flows to unstructured meshes, enabling simulations in complex geometries with the aim of applying *local dynamic* **AMR** as a further algorithm improvement both in efficiency and accuracy. The proposed approach does not involve fundamental changes in the idea of the hybrid approach between the

Level Set and Front Tracking methods. However, the method requires significant modifications in order to support arbitrary unstructured meshes, which enables the new algorithm to deal with simulations involving complex geometries for the first time. Algorithmic extensions required to support the Level Set / Front Tracking method on unstructured meshes are not straightforward, and the difference in the complexity and efficiency compared to structured mesh algorithms is emphasized throughout the algorithm description in chapter 9. Surprisingly, the increase in algorithmic complexity does not hinder the efficiency of the method in the sense of applicability to two-phase DNS. The LENT method supports both two- and three-dimensional calculation and is parallelized using the domain decomposition approach.



---

DEVELOPED METHOD

---

The proposed Level Set / Front Tracking (**LENT**) method is similar to the methods proposed by Cenicer0s et al. [22] and Shin and Juric [125]. However, significant differences are present in the definition of the front, the algorithm used for the front reconstruction as well as in the algorithm used to evolve the front. The description of the hybrid Level Set / Front Tracking method algorithm is provided for three-dimensional calculations, as they are more complex than the calculations in two dimensions. However, no change of the algorithm is necessary whatsoever in order to support pseudo-2D calculations, where a 2D discretized domain  $\Omega_h$  consists of a single layer of cells

Unlike for the LCRM method of Shin and Juric [125], and the LEFT method of Cenicer0s et al. [22], where the fronts are represented as unordered sets of disconnected triangles, the LENT front  $\mathcal{F}$  is defined as an unordered set of  $N_{\mathcal{T}}$  *connected* triangles,

$$\mathcal{F} := \{\Delta_k : k = 1, \dots, N_{\mathcal{T}}\}, \quad (187)$$

with  $\Delta_k$  denoting an area of the triangle  $k$ . The triangles  $\mathcal{T}$  of the front  $\mathcal{F}$  are mutually connected such that for each triangle there exists a von Neumann neighborhood of triangles, i.e.

$$\mathcal{N}(\mathcal{T}_k) := \{\mathcal{T}_n : \mathcal{T}_n \cap \mathcal{T}_k \text{ is an edge of } \mathcal{T}_k\}. \quad (188)$$

This kind of topological connectivity is available for the 3D topologically changing front introduced by Jayaraman et al. [63], and can be used to compute the topological changes of the front directly, by modifying the front topology. However, the LENT algorithm in the current form does not resort to topological front information to handle interface coalescence and breakup, since the topological changes of the front are automated by the iso-surface reconstruction. The connectivity of the front triangles is retained nevertheless, since it allows a possible future extension of the algorithm to take into account transport processes occurring on and across the front as well as surface tension force calculation as developed for the Front Tracking method.

Note that the front triangles  $\mathcal{T}$  may store copies of the front vertices  $v$ , or be defined as sets of indices to the set of unique front vertices  $\mathcal{V}$ . The *indirect addressing* approach (relating to a global set of points indirectly via indices) has advantages in lowering the memory usage by using unique vertices in this case, as

an entirely new front is re-created by the reconstruction. Consequently, there is no need to assure the point uniqueness by checking point equality. The global set of front vertices  $\mathcal{V}$  is used to define each front triangle  $\mathcal{T}$  as a set of indices to front vertices, i.e.

$$\mathcal{T}_k = \mathcal{T}(\Delta_k) := \{i : \mathcal{V}(i) \in \Delta_k\}. \quad (189)$$

The front triangles are defined as ordered sets of indices since the index ordering determines the computation of the triangle area normal vector. With a shorthand notation for the front vertex as

$$\mathbf{v}_{\mathcal{T},i} = \mathcal{V}(\mathcal{T}(i)), \quad (190)$$

the triangle area normal area vector is defined as

$$\mathbf{n}_{\mathcal{T}} = 0.5(\mathbf{v}_{\mathcal{T},1} - \mathbf{v}_{\mathcal{T},0}) \times (\mathbf{v}_{\mathcal{T},2} - \mathbf{v}_{\mathcal{T},0}). \quad (191)$$

Having defined the front used by the LENT algorithm, the algorithm itself can be described. The LENT algorithm is outlined by algorithm 20. The sub-algorithm details and semantics of algorithm 20 are described in the remaining sections. Those sub-algorithms of the hybrid Level Set / Front Tracking method that are in common with the geometrical VoF method are described in part I of this work and only referenced in this part.

---

**Algorithm 20** Level Set / Front Tracking (LENT)

---

Initialize the signed distance fields.	▷ section 9.1
<b>while</b> simulation time <b>do</b>	
Reconstruct the front as an iso-surface.	▷ section 9.2
Compute the marker field.	▷ section 9.3
Evolve the front.	▷ section 9.4
Update signed distance fields.	
<b>end while</b>	

---

### 9.1 COMPUTING THE SIGNED DISTANCE FIELDS

Two signed distance fields are required by the LENT algorithm: distances between the cell centers and the front ( $\phi_c$ ) and distances between the cell corner points and the front ( $\phi_p$ ). The shortest signed distance between a point in space  $\mathbf{x}$  and the front  $\mathcal{F}$  is calculated as

$$\phi(\mathbf{x}, \mathcal{F}) = \begin{cases} \phi(\mathbf{x}, \mathbf{x}_{\mathcal{T}_N}), & \mathbf{r}_{x, \mathcal{T}_N} \text{ intersects } \mathcal{T}_N \\ \phi(\mathbf{x}, \mathbf{v}_{\mathcal{T}_N}), & \text{otherwise.} \end{cases} \quad (192)$$

The ray  $\mathbf{r}_{x, \mathcal{T}_N}$  is the nearest distance ray from  $\mathbf{x}$  to  $\mathcal{T}_N$  given by the triangle plane. Equation (192) defines the distance between the front and any point as the distance between the front triangle which is nearest to the point ( $\mathcal{T}_N$ ) and the point ( $\mathbf{x}$ ).



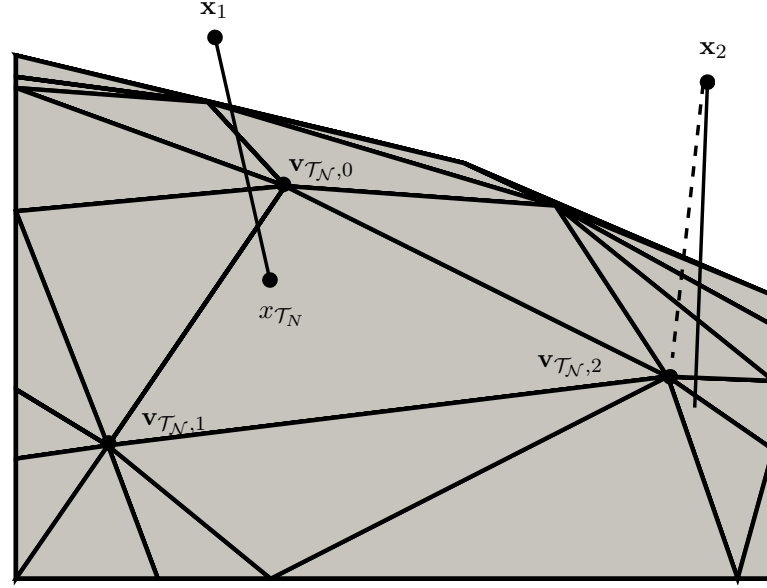


Figure 40: Signed distance calculation computed separately for each of the two example points ( $\mathbf{x}_1$ ,  $\mathbf{x}_2$ ) and the front  $\mathcal{F}$ . For the point  $\mathbf{x}_2$ , the shortest ray to  $\mathcal{T}_N$  does not intersect with the triangle, so the signed distance is computed using the closest triangle point  $\mathbf{v}_{\mathcal{T}_N,2}$ .

The nearest triangle to a point is defined as the front triangle whose distance to the front computed with equation (192) is minimal.

Following equation (192) and figure 40, if a ray projected from the point  $\mathbf{x}$  to the nearest triangle  $\mathcal{T}_N$  in the opposite direction of the triangle normal  $\mathbf{n}_{\mathcal{T}}$  intersects the nearest triangle (as it does for point  $\mathbf{x}_1$ ), the signed distance between the point  $\mathbf{x}$  and the triangle intersection point  $\mathbf{x}_{\mathcal{T}_N}$  is taken as the signed distance between the point  $\mathbf{x}$  and the whole front  $\mathcal{F}$ . However, as in the case of the point  $\mathbf{x}_2$ , it may happen that the intersection between the ray and the nearest triangle does not fall within the nearest triangle. In this case the distance between  $\mathbf{x}$  and  $\mathcal{F}$  is computed using equation (192) as the minimal distance between  $\mathbf{x}$  and *the nearest triangle vertex*. Computing the signed distance between a point and a triangle is an often encountered algorithm in computational geometry. A similar algorithm to the one presented here uses parametrized triangle and is described by Schneider [121].

Computing the signed distance fields  $\phi_p$  and  $\phi_c$  using equation (192) naively would involve finding the nearest triangle using the entire front, for every cell center and every mesh point. Such signed distance field calculation algorithm would have a computational complexity of  $O(N_{\mathcal{F}}N_p + N_{\mathcal{F}}N_c)$ , where  $N_{\mathcal{F}}$  is the number of front triangles,  $N_p$  is the number of mesh points and  $N_c$  is the number of mesh cell centers. Such computational complexity might render the algorithm unusable for simulation cases involving a large number of cells, as is often the case for two-phase DNS. For this reason, the front/mesh communication algorithm is modified in order to increase the computational efficiency substantially and increase the computational

performance of the **LENT** method on unstructured meshes. Additionally, a non-standard narrow-band calculation for the signed distance field (and related fields) is introduced that further decreases computational costs.

In order to compute the minimal distance between a point and a triangle, the point needs to be located somewhere within the mesh and for this purpose unstructured mesh search operations are used.

### 9.1.1 *Unstructured mesh search operations*

The Front Tracking method is in its basis an Immersed Boundary method with a topologically changing boundary. A review on the immersed boundary method is given by Mittal and Iaccarino [87] and the method was originally developed by Peskin [102]. There are many publications available on the Immersed Boundary method and the review work of Mittal and Iaccarino [87] is a good starting point for further investigations. The problem of efficiently and accurately tracking the immersed boundary has already been addressed in detail on structured meshes. The required search operations on unstructured meshes are much more complex and require careful attention.

The position of the front determines the phase properties and positions in a two-phase DNS problem in the form of computing Eulerian fields on the background Eulerian mesh. In general, the motion of the front is governed by the velocity resulting from the solution of a Navier-Stokes equation system in single-field formulation on the background Eulerian mesh. For this purpose there needs to exist a communication function between the front and the mesh (e.g. relating  $\mathcal{T}$  to  $\mathbf{p}$ ). On structured meshes, as described by Tryggvason et al. [139], the location of a front vertex in a specific mesh cell can be computed in a very straightforward way as

$$i = FLOOR\left(v_x \cdot \frac{N_x}{L_x}\right) = \lfloor v_x \cdot \frac{N_x}{L_x} \rfloor, \quad (193)$$

where  $i$  is the structured mesh cell index,  $v_x$  is the front vertex coordinate,  $N_x$  is the number of volumes used to discretize the domain and  $L_x$  is the length of the domain, in the direction of the  $x$  axis. This equation can be modified easily to take into account block refinement of a structured mesh as shown by Cenicerós et al. [23].

On unstructured meshes, locating a front vertex with respect to the mesh cell cannot be done using equation (193), because of the irregularity of the mesh cells. To this purpose, a combination of an octree search algorithm 21 and a known-neighbor search algorithm 22 is applied. Both algorithms combined together result in a fast search mechanism that works well with search points moving across the unstructured mesh. The point motion involved in tracking the front with the LENT algorithm is specific in the sense that the connectivity between the points and the cells can be cached and re-used and re-calculated only sporadically. These implications result in a point search algorithm that has a low computational

overhead. Both the octree search and the known-vicinity search algorithms are outlined in algorithm 21 and algorithm 22.

---

**Algorithm 21** Octree-based search

---

```

function OCTREE-SEARCH(node, point, squaredDist, nearestPoint)
  for octants of node do
    get subNode
    pointSphere = sphere around point using squaredDist
    if subNode has child nodes then
      subNodeBox = Axis-Aligned Bounding Box (AABB)(subNode)
      if subNodeBox overlaps with pointSphere then
        OCTREE-SEARCH(subNode, point, squaredDist, nearestPoint)
      end if
    else if subNode is a leaf node then
      leafNodeBox = Axis Aligned Bounding Box (node)
      if do_intersect(leafNodeBox, pointSphere) then
        minimalPoint = distant point
        for shape in node shapes do
          shapeDist, shapePoint = SIGNED DISTANCE(shape, point)
          if ||shapeDist|| < ||minimalDistance|| then
            minimalPoint = shapePoint
          end if
        end for
        nearestPoint = minimalPoint
      end if
    end if
  end for
end function

```

---

The octree data structure was first used for representing complex 3D shapes by Meagher [85] and has since then been widely used in different fields such as computational geometry, computer graphics and numerical simulations. The search algorithm based on the octree representation of complex bodies simplifies various query operations. Löhner [74] has used octrees to optimize search operations on unstructured meshes. More information on the octree data structure and related algorithms can be found in Samet [118]. The octree based search algorithm employed by the LENT method and shown in algorithm 21 relies on a regular octree which recursively partitions space into octants holding as content the input geometry. In the case of the hybrid Level Set / Front Tracking method, the input geometry is the front  $\mathcal{F}$ . However, the space partitioning is not refined based on the front geometry. The tree depth is fixed and can be prescribed and is as such not related e.g. to the triangle size. Therefore, the octree implementation presents a point of possible future algorithm improvement. Nevertheless, the test cases show promisingly short execution times.

Algorithm 22 is a heuristic approach to searching on unstructured meshes, similar to one proposed by Löhner [74]. Modifications to the algorithm of Löhner [74] have been applied in the sense of the *owner-neighbor addressing* topology of the unstructured mesh in OpenFOAM as well as special cases when the algorithm diverges. The description of algorithm 22 omits the fact that the information on

the neighboring cells is provided by the mesh, in order to simplify exposition and understanding.

Figure 41 shows schematically the steps of the algorithm 22 on a polygonal mesh. The search starts within a seed cell. From this cell, the neighboring cells are investigated. The minimal distance between the point and the center of each neighboring cell is computed. The point-in-polyhedron algorithm is a test described by algorithm 23.

---

**Algorithm 22** Known vicinity search

---

```

function KNOWN-VICINITY-SEARCH(point, cell)
  minDistance = large number
  minCell = -1
  for nextCell in neighboring cells do
    cellToNextCellDist = distance(cell, nextCell)
    if cellToNextCellDist < minDistance then
      minDistance = cellToNextCellDist
      minCell = nextCell
    end if
  end for
  if POINT-IN-POLYHEDRON(point, polyhedron(minCell)) then
    return minCell
  else
    KNOWN-VICINITY-SEARCH(p, minCell)
  end if
end function

```

---



---

**Algorithm 23** Point in convex polyhedron

---

```

function POINT-IN-POLYHEDRON(point, polyhedron)
  for polygon in polyhedron do
    polygonPoint = polygon.points()[0]
    if (point - polygonPoint) · polygonNormal > 0 then
      return false
    end if
  end for
  return true
end function

```

---

The overall search algorithm will compute the octree-based search and resort to the known-vicinity search and vice-versa, when required. This will be defined by the iso-surface reconstruction part of the LENT algorithm, which is described in the following section. In general, the combination of two proposed algorithms converges unconditionally to a cell, for such a point that lies within the flow domain, as the octree-based search is always used as a fall-back solution.

### 9.1.2 Narrow band generation and propagation

Comparing equation (193) with algorithm 21 and algorithm 22 shows that the communication between the front and the Eulerian mesh is more complicated

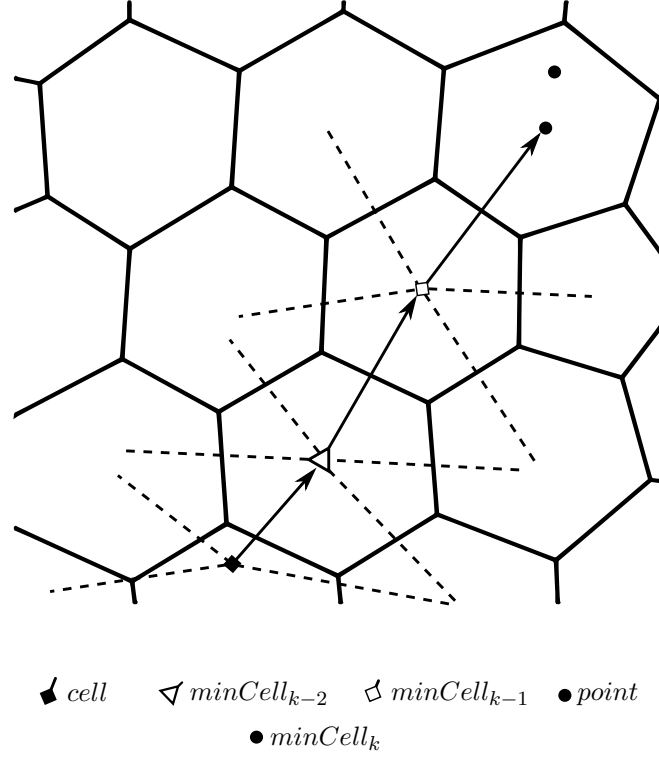


Figure 41: The known-vicinity search algorithm operating on a polygonal mesh in  $k$  iterations. The dashed lines represent the topological connections between the current cell and the neighboring cells. Of all the possible connections, only one is followed by the algorithm to the next cell and is visualized as an arrow line. The next cell is the neighbor cell whose center has the minimal distance to the point which is searched for.

for the unstructured mesh compared to (block) structured meshes. This issue is addressed by applying a modified narrow band approach. The narrow band approach was introduced by Sethian [123] for the level set method. However, the narrow band algorithm in the hybrid Level Set / Front Tracking method differs from the original one. No equations are solved in the narrow band for the signed distance field. To the contrary, the signed distance is computed directly from the front within the narrow band.

The complexity of algorithm 21 is a function of the squared radius of the search distance for a point  $\mathbf{p}$ . The larger the search radius, the more octree nodes need to be examined. The narrow band is generated indirectly, by setting the search distance for each cell. The squared search distance of a polyhedral cell is defined as

$$s_c = \frac{1}{|\mathcal{N}_c|} \sum_{i=1}^{|\mathcal{N}_c|} \|d_{c,i}\|^2, \quad (194)$$

where  $\mathcal{N}_c$  is the von Neumann neighborhood of the cell  $c$ , defined analogously as the von Neumann neighborhood of a triangle  $\mathcal{T}$  by equation (188), i.e.

$$\mathcal{N}(c) = \mathcal{N}_c := \{c_n : c_n \cap c \text{ is a face of } c\}. \quad (195)$$

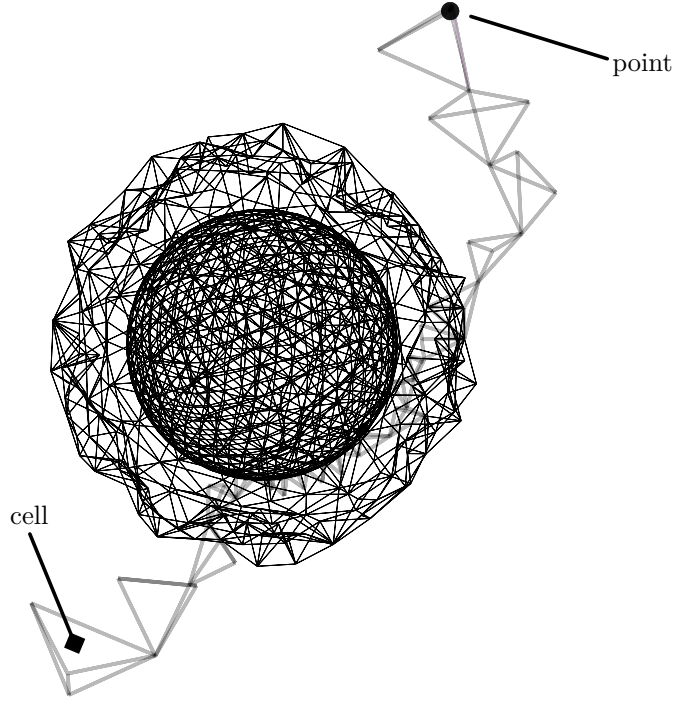


Figure 42: The known-neighborhood search algorithm operating on a tetrahedral mesh with an obstacle between the seed cell and the point in form of a sphere. Stepping cells (*minCell* cells on figure 41) build a search path between the cell and the point. The tetrahedral mesh used for this example is of low quality, which can be seen in the large discrepancy of cell dihedral angles as well as in the sizes of the cells in the search path.

The distance between two cells  $d_{c,i}$  is computed as the distance between their respective centers. The search distance for the mesh points is computed as an arithmetic average of the cell distances of the surrounding cells:

$$s_p = \frac{1}{|\mathcal{C}_p|} \sum_{i=1}^{|\mathcal{C}_p|} s_c(\mathcal{C}_p(i)), \quad (196)$$

where  $\mathcal{C}_p$  is a point-cell neighborhood of a mesh point  $\mathbf{p}$ , i.e.

$$\mathcal{C}_p(\mathbf{p}) := \{c : \mathbf{p} \in c\}. \quad (197)$$

There are no additional data structures or boundary conditions required for the narrow band. Furthermore, the complexity of the initial global calculation is never repeated: algorithm 21 is executed for the whole computational domain only once, as a preprocessing step. All the other invocations, provided that algorithm 22 fails in the first place, will not be global - they will start within an octree node defined by a previously cached cell.

The initial field required for the initialization needs to be preset with a large positive value  $\phi_L$  as described by Sethian [123] and Shin and Juric [125], among others. Once the narrow band is computed, there will be a discrepancy between the cells with negative values of  $\phi$  on one side and a large positive value  $\phi_L$  on the

other side. A mesh propagation algorithm is applied to propagate the narrow band information, described with algorithm 24. The naive narrow band propagation loops over mesh elements (edges for a point field, and faces for cell field), tests if there is a mesh element pair  $(O, N)$  for which one value of  $\phi$  is negative and the other is  $\phi_L$ . The index pair  $(O, N)$  corresponds to the face owner and face neighbor cell, when the elements in question are mesh cells. For mesh edges, the pair  $(O, N)$  represents the first and the second point of the edge.

---

**Algorithm 24** Naive narrow band propagation

---

```

currentItem = mesh item    ▷ It can be a cell, triangle, edge, depending on the mesh
                             type.
jumpItem = currentItem
while jumpItem > 0 do
    jumpItem = -1
    if ( $\phi(O) < 0$ ) and ( $\phi(N) == \phi_L$ ) then
        jumpItem = currentItem
         $\phi(O) = -1 \cdot \phi(O)$ 
    end if
    if ( $\phi(N) < 0$ ) and ( $\phi(O) == \phi_L$ ) then
        jumpItem = currentItem
         $\phi(N) = -1 \cdot \phi(N)$ 
    end if
end while

```

---

It is important to note that the narrow band generation and propagation, as well as the search distance, all rely on compact cell stencils, allowing for an efficient communication when parallelizing the algorithm using the domain decomposition approach.

### 9.1.3 Narrow band properties

The implementation of the narrow band approach is not a standard one, where the subset of cells is used to represent a field. One might argue that the reduction in the computational costs is not justified since the field operations will be performed for the bulk of both phases. Exactly the opposite has been observed: profiling of the algorithm shows that the computational bottleneck lies not in the narrow band propagation and field operations for the LENT algorithm. The majority of the computation is performed by the search operations, which is expected for unstructured meshes.

## 9.2 RECONSTRUCTING THE FRONT AS AN ISO-SURFACE

Once the signed distance fields  $\phi_p$  and  $\phi_c$  are computed, each cell of a mesh can be used to polygonize the surface implicitly defined with the signed distance fields, as an iso-surface. The polygonization is very similar to the algorithms used first in the context of two-phase DNS by Shin and Juric [125] and then later by Ceniceros et al. [22]. The polygonization algorithm was used much earlier in the field of

computer graphics, by Bloomenthal [19]<sup>1</sup>. For the LENT algorithm, tetrahedral decomposition is performed with barycentric triangulation of cell faces, which ensures that the iso-surface can be triangulated in cells of different shapes.

The LENT algorithm relies on the iso-surface reconstruction as proposed by Treece et al. [137]. Shin and Juric [125] have proposed the use of this algorithm with the tetrahedral decomposition of Bloomenthal [19] in order to ensure topological regularity of the front for their LCRM method. However, as the implementation supports unstructured meshes, a barycentric triangulation of each mesh cell is used in order to allow the iso-surface reconstruction algorithm to handle different cell shapes. Simplification of the front is not considered at this point (also described by Treece et al. [137] and later by Shin and Juric [126]), and is left as a possible future enhancement.

As the iso-surface reconstruction is a widely used algorithm, its description is not repeated for the sake of conciseness. Figure 43(a) and figure 43(b) show the difference between the oriented tetrahedral decomposition proposed by Bloomenthal [19] and used by Shin and Juric [125] and the barycentric tetrahedral decomposition that the LENT algorithm relies on.

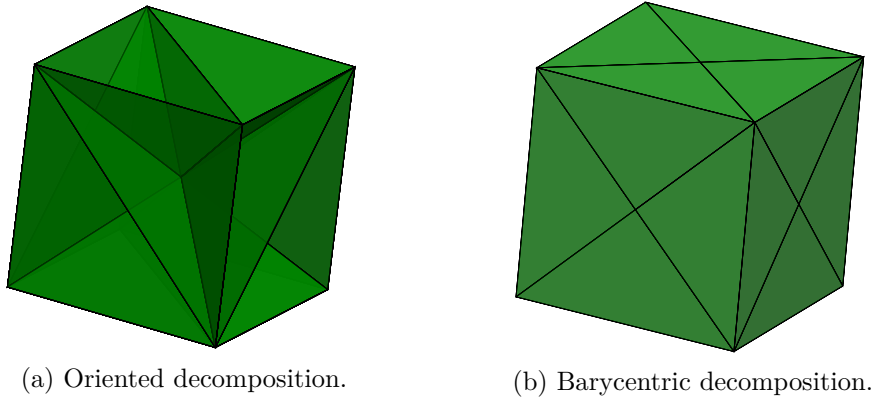


Figure 43: Different approaches to tetrahedral decomposition.

### 9.3 COMPUTING THE MARKER FIELD

Equation (198) shows the model used for computing the marker field from the signed distance field, where  $h_c$  is the cell-centered approximation of a marker field:

$$h_c = \begin{cases} 0 & \text{if } \phi_c < -\sqrt{s_c}, \\ 0.5 \left[ 1 + \frac{\phi_c}{\sqrt{s_c}} + \frac{1}{\pi} \sin \left( \frac{\pi}{\sqrt{s_c}} \right) \phi_c \right] & \text{if } |\phi_c| < \sqrt{s_c}, \\ 1 & \text{if } \phi_c > \sqrt{s_c}. \end{cases} \quad (198)$$

<sup>1</sup> This is the oldest publication that describes an iso-surface polygonization algorithm known to the author.



An alternative sharp marker field model could be computed using

$$h_c = \begin{cases} 0 & \text{if } \phi_c < -\sqrt{s_c}, \\ 0.5 & \text{if } -\sqrt{s_c} \leq \phi_c \leq \sqrt{s_c}, \\ 1 & \text{if } \phi_c > \sqrt{s_c} \end{cases} \quad (199)$$

The difference between equation (198) and equation (199) is visible on coarse meshes in the case when a wider narrow band is used. The narrow band width determines the width of the spatial interval within which the signed distance field varies in values from  $-\phi_L$  to  $\phi_L$ . The phase properties are modeled using the marker field in the same way as for the geometrical **VoF** method (chapter 2). Therefore, a wider narrow band with the harmonic marker field model equation (198) results in a smoother transition region between two phases.

Marker field models given by equations (198) and (199) can be replaced by any algorithm that computes the marker field from the front, such as the **SMCI** algorithm 4. A fast and second-order accurate alternative marker field model was proposed recently by Detrixhe and Aslam [31], based on the polynomial approximation of the fill level of each tetrahedron in the cell triangulation.

Since the marker field is in effect the volume fraction field of the Front Tracking method, errors in the marker field computation cause errors in volume conservation. Even if it is assumed that the evolution of the front is computed along analytical trajectories given by a velocity  $\mathbf{u}(\mathbf{x}, t)$  without any temporal integration errors or the deviation from the condition  $\nabla_c \cdot \mathbf{u}(\mathbf{x}, t) = 0$ , any error in computing the fill level of a cell based on the topology of the front causes a local volume conservation error. The model proposed by Detrixhe and Aslam [31] is second-order accurate, however does not exactly conserve volume neither locally, nor globally. Therefore, the feasibility of using exact intersection algorithms such as **SMCI** algorithm 4 should be investigated in more detail, possibly leading to a hybrid Level Set / Front Tracking / Volume-of-Fluid approach.

## 9.4 EVOLVING THE FRONT

The evolution of the front is governed by a kinematic equation:

$$\partial_t \mathbf{v} = \mathbf{u}_v(t), \quad (200)$$

where  $\mathbf{v}$  is the front vertex position vector and  $\mathbf{u}_v$  are the velocities of the vertices  $v$  of the front  $\mathcal{F}$ . The solution of equation (200) is approximated using a first-order accurate Euler scheme, as well as a second-order Backward Differencing Scheme (**BDS**). Applying a second-order accurate modified Euler scheme (Hoffman and Frankel [57]) is also possible, but it would either involve invoking the pressure velocity coupling algorithm just to obtain the velocity used by the corrector step, which then gets thrown away by the overall solution algorithm, or decouple the transport of the marker field from the solution of the rest of the **NS** system. For this reason, the single-step explicit Taylor integration is performed for the cell displacements, that are then interpolated to the front vertices, as described in detail

in section 6.3.1. The sub-algorithm of integrating cell center displacements and their interpolation to cell corner points of the geometrical VoF method is equivalent for the LENT method. However, for better clarity, the necessary computations are also described here.

In order to evolve the front  $\mathcal{F}$ , equation (200) requires a known velocity of each front vertex  $\mathbf{v}$ . For the verification of the front evolution,  $\mathbf{u}_v$  is prescribed by an explicit function whose definition depends on the chosen verification case. The velocity of the front vertices  $\mathbf{u}_v$  is not prescribed directly, although this would result in significantly smaller error values. Instead, the cell-centered velocities are interpolated to front vertices, since the cell-centered velocities are used by the LENT algorithm when it is coupled with the flow solution algorithm. This approach enforces the verification process for the front evolution to take into account the interpolation errors between the Eulerian mesh and the front, ensuring an early detection of problems in this part of the algorithm development.

---

**Algorithm 25** Mesh-to-front interpolation

---

```

for  $\mathcal{M}_c, \mathcal{T}$  do
   $c = \mathcal{M}_c(\mathcal{T})$ 
  for  $\mathcal{T}, \mathbf{v}$  do
    vertexCell =  $c$ 
    if not POINT-IN-POLYHEDRON( $\mathbf{v}$ , polyhedron( $c$ )) then
      vertexCell = KNOWN-VICINITY-SEARCH( $\mathbf{v}$ ,  $c$ )
    end if
    INTERPOLATE-VERTEX-VELOCITY( $\mathbf{v}$ ,  $c$ )
  end for
end for

```

---

To compute the velocity of the vertex  $\mathbf{v}$ , a triangle-cell map  $\mathcal{M}_c(\mathcal{T})$  is defined as

$$\mathcal{M}_c(\mathcal{T}) := c \quad \text{if } \mathcal{T} \cap c \neq \emptyset. \quad (201)$$

When  $\mathcal{F}$  is reconstructed, there will be only one cell  $c$  in  $\mathcal{M}_c(\mathcal{T})$  defined by equation (201) since all triangles will be completely contained within the cell  $c$ . As the front evolves,  $\mathcal{M}_c(\mathcal{T})$  needs to be updated. This is performed by the search algorithms described in section 9.1.1, which modify  $\mathcal{M}_c(\mathcal{T})$  in the following way:

$$\mathcal{M}_c(\mathcal{T}) := c \quad \text{if } c \text{ is nearest to } \mathcal{T}. \quad (202)$$

$\mathcal{M}_c(\mathcal{T})$  is always kept up-to-date by the LENT algorithm and it is used to interpolate  $\mathbf{u}_v$ . Note, however, that the cell  $c$  given by equation (202) does not need to be the cell that contains  $\mathbf{v}$ . This happens when  $\mathcal{F}$  evolves over a longer period of time without having been reconstructed. In this case, algorithm 22 is used to locate the cell actually containing  $\mathbf{v}$ .

A barycentric (volume-weighted) interpolation can be used for interchanging field values between the front and the mesh. The mesh point values are interpolated

using the **IDW** interpolation from the cell centers to the cell points  $p$ , shown here for a generic property  $\Psi$ :

$$\Psi_p = \sum_{pc \in \mathcal{M}_c(p)} w_{pc} \Psi_{pc}, \quad (203)$$

where  $\mathcal{M}_c(\mathbf{p})$  is defined as

$$\mathcal{M}_c(p) := \{c : (p \cap c) = p\}, \quad (204)$$

and the interpolation weights  $w_{pc}$  are the inversed distance weights, i.e.

$$w_{pc} = \frac{\|\mathbf{p} - \mathbf{x}_c\|^k}{\sum_{\tilde{p}c} \|\mathbf{p} - \mathbf{x}_c\|^k}. \quad (205)$$

$k = -1$  is used for the results shown in chapter 12, although other values are applicable, as well as other interpolation methods. Having calculated the mesh point values, the cells are decomposed into tetrahedra as shown in figure 43(b), and the point  $\mathbf{p}$  is placed within a tetrahedron  $\mathcal{T}_e$  of the cell  $c$ . The point  $\mathbf{p}$  further divides  $\mathcal{T}_e$  into four tetrahedra whose volumes are then taken as weights for barycentric interpolation given as

$$\Psi_v = \sum_{tp} \Psi_{tp} w_{tp}, \quad (206)$$

where  $tp$  is the point of the tetrahedron  $\mathcal{T}_e$ , and  $w_{tp}$  is the volume fraction of the corresponding sub-tetrahedron. Algorithm 25 describes the overall interpolation procedure. In case there is a property of the front  $\mathcal{F}$  interpolated to the Eulerian mesh, the procedure stays the same, since the map  $\mathcal{M}_c(\mathcal{T})$  can be used directly for the reversed communication direction, as described by Tryggvason et al. [139].



## **Part IV**

### **Results and outlook**



---

INTRODUCTION

---

Part **IV** covers the verification of both the geometrical **VoF** method and the hybrid Level Set / Front Tracking method method. The solutions from both proposed methods are compared with results from other state of the art methods. In this text, the term verification denotes a process of comparing the errors of an algorithm implementation to an exact solution. For each listed verification case used for interface motion an exact solution is available.

Verification is extended in this work to include the computational costs of the method or a sub-algorithm, since there may be cases when the computational costs increase so severely compared to another method, that the improved order of accuracy still does not justify the use of the more accurate method. The computational costs are reported as absolute values in seconds. Since those vary dependent on the used computer architecture, the computer architecture used for each test is specified. To avoid repetition when specifying used computer architectures, they are categorized in table 3.

Times reported in tables are the average execution time  $Te$  and average reconstruction time  $Tr$ . The average advection time is defined as

$$Te = \frac{1}{Nt} \sum_{i=1}^{Nt} Tp_i + Ts_i + Tc_i + Tr_i \quad (207)$$

where:

- $Nt$  is the number of time steps,
- $Tp_i$  is the time used to compute the phase volumes,
- $Ts_i$  is the time used to update the  $\alpha$  field,
- $Tc_i$  is the time used to remove wisps and redistribute over/undershoots,
- $Tr_i$  is the time used to reconstruct the interface

The average reconstruction time  $Tr$  is defined as

$$Tr = \frac{1}{Nt} \sum_{i=1}^{Nt} Tr_i. \quad (208)$$

It is important to note that the time required to write the **PLIC** interface polygons to the hard drive is not a part of the measurement for the geometrical **VoF** method,

Architecture	
I0	
CPU	
	vendor_id : GenuineIntel
	cpu family : 6
	model : 60
	model name : Intel(R) Core(TM) i7-4710HQ CPU @ 2.50GHz
	stepping : 3
	microcode : 0x17
	cpu MHz : 3351.287
	cache size : 6144 KB
Compiler	
version	g++ (GCC) 6.2.1 20160830
optimization flags	-std=c++14, -O3
I1	
	vendor_id : GenuineIntel
	cpu family : 6
	model : 63
	model name : Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz
	stepping : 2
	microcode : 0x2b
	cpu MHz : 2300.000
	cache size : 15360 KB
Compiler	
version	g++ (GCC) 4.9.4
optimization flags	-std=c++14, -O3

Table 3: Computer architectures used for the verification cases.

because this method does not require this information for calculation or for the restarting of a previous calculation.

When multiple methods are implemented within a single software framework, the objectivity of the verification process is significantly reinforced, especially concerning computational efficiency. Of course, it is important to remember that what is being compared are the *method implementations* and not the methods themselves. An error in the implementation of a two-phase **DNS** method may cause severe computational expenses, even though the method itself might be computationally efficient. To increase the objectivity, the execution times are reported along with the details on the computer architecture used to obtain the solutions. Verification of the interface motion is performed by moving the interface with prescribed explicit velocity functions.

As in all the literature referenced throughout this thesis, the **CFL** is defined as a *domain global number*: the time step is not adaptively adjusted to account either



for the temporally varying velocity field, or the presence of the interface to enable comparison of results.

Three types of errors are measured for the verification cases: the volume conservation error  $E_v$ , the geometrical error  $E_g$  and the normalized geometrical error  $E_n$ . The volume conservation error is defined as

$$E_v = \frac{|\sum_c V_c \alpha_c(t) - \sum_c V_c \alpha_c(t_0)|}{\sum_c V_c \alpha_c(t_0)}, \quad (209)$$

where  $t_0$  is the start time of the simulation. Verification cases used for the interface advection all have exact solutions. For some cases, the interface velocity is multiplied by a  $\cos(\frac{\pi t}{T})$  term that switches the sign of the velocity after  $\frac{T}{2}$ , so that the interface is brought back to its original configuration. For the translation and rotation cases, the exact values of  $\alpha_c$  are known because the exact position of the rotated or translated interface is known at any point in time, and the interface does not undergo any deformation. Therefore, the geometrical error is always computed against the exact volume fraction field, and not one given by a calculation on a finer mesh, or by another numerical method. The geometrical error  $E_g$  is defined as

$$E_g = \sum_c V_c |\alpha_c(t^e) - \alpha_c^e|, \quad (210)$$

where  $\alpha_c^e$  is the exact volume fraction field and  $t^e$  is the time at which it is expected that the interface obtains the exact configuration. The normalized geometrical error is simply the geometrical error  $E_g$  normalized with the total volume

$$E_n = \frac{E_g}{\sum_c V_c \alpha_c^e}. \quad (211)$$

When a numerical method is used to transport a discontinuity, numerical instability ensues: transporting the volume fraction field  $\alpha_c$  by such a method causes  $\alpha_c(t) \notin [0, 1]$  for some  $c$ . This *numerical boundedness error* is defined as

$$Eb = \max(0, \max(0, \max_c(\alpha_c - 1)), \max_c(0 - \alpha_c)). \quad (212)$$



---

## GEOMETRICAL VOF METHOD

---

To verify the geometrical **VoF** method, the volume fraction field needs to be initialized. New algorithms proposed for the volume fraction initialization are covered in detail in section 6.1. The **CCI** initialization algorithm 3 is used for 2D verification cases since its complexity makes it computationally expensive for 3D cases. Figure 44 shows the cylindrical pseudo-2D mesh used to initialize circular interfaces in the test cases covered in this chapter. The pseudo-2D mesh is generated as a prismatic layer of 3D mesh cells because 2D geometrical calculations are performed by disregarding a 3D coordinate: a practice used in the OpenFOAM computational framework where the proposed geometrical **VoF** method is implemented. Additional to the reduction of complexity of the mesh intersection algorithm by **AABB** intersection tests, the absolute computational cost is reduced in figure 44 by adapting the mesh near the interface. In this case, the interface is resolved by  $1.5e03$  elements. The mesh in figure 44 was generated with the *cfMesh* meshing tool. Information required for generating the mesh of the cylinder is available in appendix B.

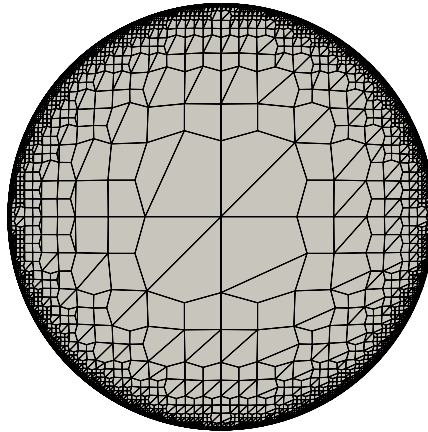


Figure 44: Cylindrical mesh used for two-dimensional verification cases.

### 11.1 TRANSLATION

The interface translation test was introduced by Rider and Kothe [111] to test the error introduced by the Eulerian flux-based approach of the geometrical **VoF** method in the interface shape. Later, Harvie and Fletcher [53] have extended this testcase to separate the influence of the geometrical reconstruction errors. As the reconstruction is driven by mesh quality for the hybrid Level Set / Front

Tracking method, the translation test case would not cause any reconstructions to be performed. This test case therefore does not directly apply to the **LENT** method.

As Harvie and Fletcher [53] state, the reconstruction errors introduce an error in the overall solution since they introduce an error in the geometrical approximation of the interface. Therefore, the overall error in the advection should decrease with a decreased number of executed reconstructions. This relationship connects the reconstruction error with the  $CFL$  condition: the influence of the reconstruction error to the solution will be smaller for larger  $CFL$  numbers, as less time steps are required to reach the end of the simulation time  $T$ . However, with the decrease of  $CFL$  number, the reconstruction error should remain stable, otherwise the method could not be used with small time steps, that are often required when solving a single-field Navier-Stokes two-phase equation system.

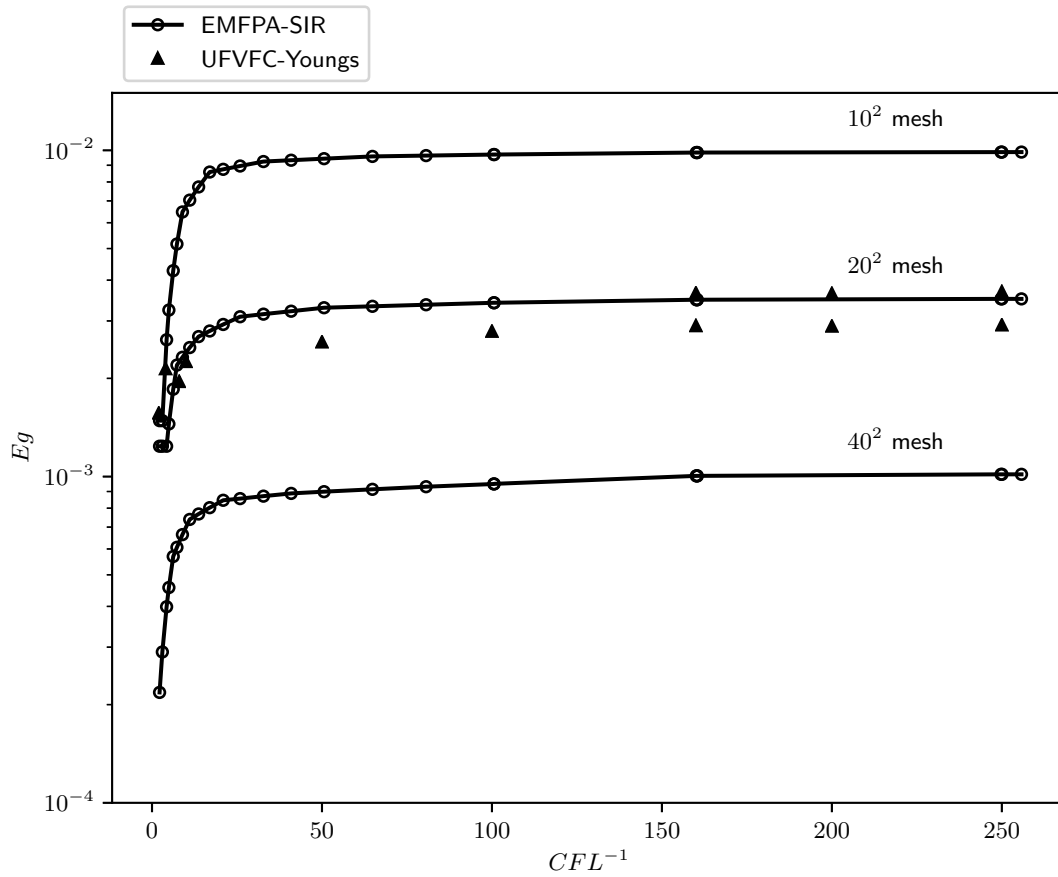


Figure 45:  $E_g$  geometrical error as a function of  $CFL^{-1}$  for the two-dimensional translation case. Results are obtained with the **UFVFC**-Youngs combination using the Taylor integration for the cell displacements, **IDW** point interpolation, **TBDS** flux integration, and pyramid flux volume correction.

A representative solution for the 2D translation has been presented by López et al. [75]. The test consists of a circular interface of radius  $R = 0.2$  initially placed at  $\mathbf{c}_R = (0.25, 0.25, 0)$  and translated with the constant velocity  $\mathbf{U} = (1, 1, 0)$  for a time  $T = 0.5$  in a square domain of dimensions  $1 \times 1$ . López et al. [75] have used a spline-based interface reconstruction method to ensure the second-order of accuracy of their Edge-Matched Flux Polygon Advection and Spline Interface

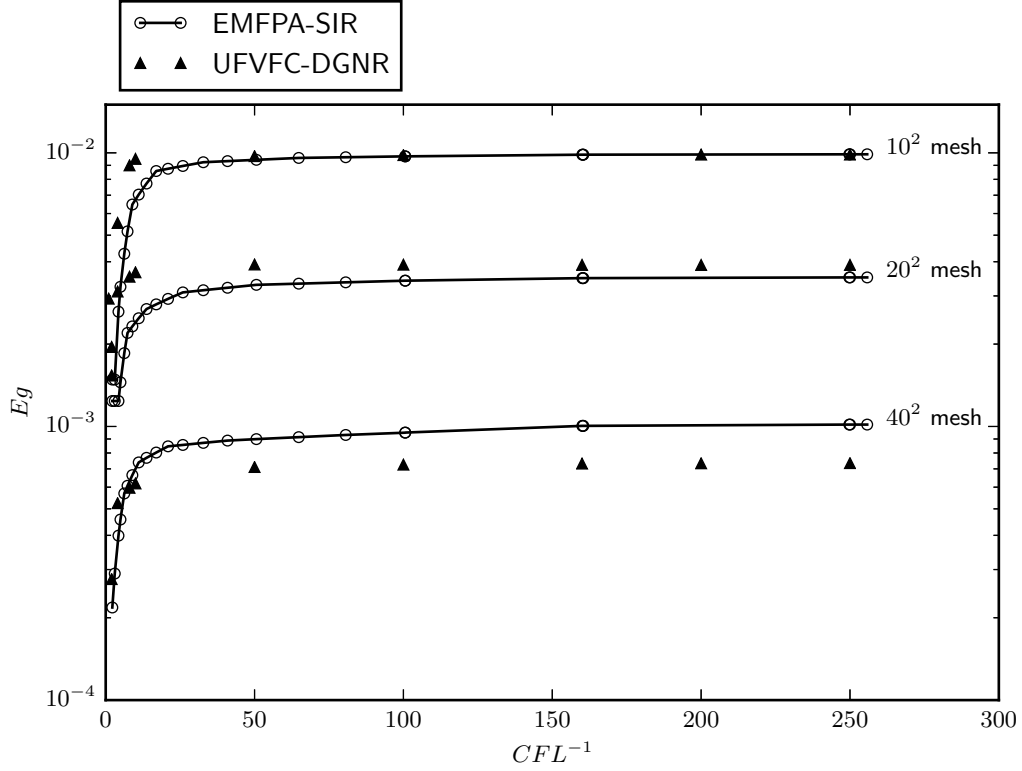


Figure 46:  $E_g$  geometrical error as a function of  $CFL^{-1}$  for the two-dimensional translation case. Results are obtained with the **UFVFC-DGNR** combination with 2 reconstruction steps, using the Taylor integration for the cell displacements, **IDW** point interpolation, **TBDS** flux integration, and pyramid flux volume correction.

Reconstruction (**EMFPA-SIR**) method. At the time of this writing this method does not have its direct 3D equivalent, however it produced very accurate results and is therefore used here for comparison with the geometrical **VoF** method.

When Youngs' algorithm is used together with **UFVFC**, the reconstruction errors remain stable with decreasing **CFL** as shown in figure 45. However, because the increase in the mesh resolution resolves the sharp change in the  $\alpha$  field, the **UFVFC**-Youngs algorithm combination does not converge fast with increased mesh resolution.

Results shown in figure 46 confirm that the **UFVFC-DGNR** combination delivers a stable reconstruction error with the decrease of the **CFL** number. The absolute accuracy is slightly worse when compared to **EMFPA-SIR** on very coarse resolutions with only  $10^2$  and  $20^2$  volumes, which is to be expected because the **DGNR** reconstruction algorithm relies on gradient reconstruction using directional derivatives of the distance function, instead on the higher order B-spline reconstruction. However, **DGNR** shows better results and above second-order convergence between meshes  $20^2$  and  $40^2$ , which makes it a very promising alternative. Furthermore, the **DGNR** algorithm is dimension-independent: it can be used in both two-and-three dimensions without any modification.

		CFL = 0.25	CFL = 0.5	CFL = 1.0
CVTNA +	32.0	2.09e-03	1.53e-03	1.37e-03
direction-split	-	1.95	2.01	1.95
-	64.0	5.41e-04	3.79e-04	3.54e-04
-	-	2.03	2.07	2.13
-	128.0	1.30 4	9.03e-05	8.08e-05
Youngs +	32.0	2.25e-03	1.50e-03	9.90e-04
PCFSC unsplit	-	1.93	1.90	1.62
-	64.0	5.90e-04	4.00e-04	3.20e-04
-	-	1.32	1.11	1.18
-	128.0	2.37e-04	1.86e-04	1.40e-04
CVTNA +	32.0	2.39e-03	2.03e-03	1.64e-03
PCFSC unsplit	-	2.08	2.33	2.16
-	64.0	5.65e-04	4.00e-04	3.67e-04
-	-	2.10	2.03	1.76
-	128.0	1.30e-04	9.83e-05	1.08e-04

Table 4: Reference  $E_g$  errors of the 2D rotation test done by Liovic et al. [73]. The numbers  $N = 32, 64, 128$  represent different mesh resolutions  $N^2$ . Numbers inbetween rows define the orders of convergence.

## 11.2 ROTATION

The test consists of a circular interface of radius  $R_{r2D} = 0.15$ , centered at  $\mathbf{c}_{r2D} = (0.5, 0.75, 0)$  and revolved around the axis  $\mathbf{a}_{r2D} = (0.5, 0.5, 0)$  for a single complete revolution with the angular velocity  $\omega_{r2D} = (0, 0, 1)$  and  $CFL = 0.25, 0.5, 1$  in a domain of dimensions  $1 \times 1$ .

The 2D rotation verification case with a circle was performed by Liovic et al. [73] to test their PCFSC-CVTNA scheme. Their results are shown in table 4. López et al. [75] have also verified their EMFPA method using SIR and LVIRA for the reconstruction, and their results are shown in table 5.

As expected and reported by López et al. [75] and Liovic et al. [73], the reconstruction algorithm determines the order of convergence for the 2D rotation step. For this case  $\partial_t \mathbf{u} = \mathbf{0}$  so the choice of the temporal scheme for the integration of cell displacements as well as fluxes has no influence in the accuracy and convergence of the solution. Results computed with the IDW interpolated point velocities in table 6 show the decrease of the order of convergence caused by the use of the Youngs' reconstruction algorithm.

The Swartz reconstruction method with 10 normal correction steps produces a stable second-order convergence, and an absolute accuracy comparable to the combination EMFPA-LVIRA of López et al. [75]. Its results are shown in table 7. Below 10 normal correction steps, the second-order convergence cannot be obtained.

Of the two new reconstruction methods, the CCNR reconstruction brings the most promising results in table 8, with 3 reconstruction steps. The errors obtained with the CCNR reconstruction are better than those reported by Liovic et al. [73]

Advection Reconstruction	-	Rider and Kothe Puckett	EMFPA Puckett	EMFPA SIR
32	E	1.61e-03	1.42e-03	8.68e-04
-	O	2.19	2.08	1.76
64	E	3.54e-04	3.37e-04	2.57e-04
-	O	1.98	2.08	2.22
128	E	8.95e-05	7.96e-05	5.52e-05

Table 5: *Reference* $E_g$  errors (E) and orders of convergence (O) of the 2D rotation test by López et al. [75], with  $CFL \approx 0.5$  corresponding to 201 time steps for the  $32 \times 32$  case.

CFL	N	Ev	Eb	Eg	O(Eg)	Te	Tr
0.25	32	9.82e-16	0.0	1.82e-03	1.82	1.50e-02	1.28e-03
	64	4.66e-15	0.0	5.16e-04	1.18	3.08e-02	3.07e-03
	128	1.82e-14	0.0	2.27e-04	-	7.08e-02	7.20e-03
0.50	32	8.59e-16	0.0	1.01e-03	1.45	1.59e-02	1.42e-03
	64	4.66e-15	0.0	3.68e-04	1.04	3.22e-02	3.06e-03
	128	1.80e-14	0.0	1.79e-04	-	7.40e-02	7.19e-03
1.00	32	1.10e-15	0.0	8.27e-04	1.59	1.70e-02	1.14e-03
	64	4.79e-15	0.0	2.75e-04	1.14	3.58e-02	3.18e-03
	128	1.80e-14	0.0	1.24e-04	-	8.24e-02	7.50e-03

Table 6: Results of the 2D rotation test case with the Youngs reconstruction, Taylor integration for the cell displacements, IDW point interpolation, TBDS flux integration, and pyramid flux volume correction. Times are reported for the execution on a single process on the I0 architecture.

		Ev	Eb	Eg	O(Eg)	Te	Tr
CFL	N						
0.25	32	1.23e-15	0.0	1.85e-03	1.98	1.56e-02	3.39e-03
	64	4.66e-15	0.0	4.71e-04	2.03	3.38e-02	7.73e-03
	128	1.82e-14	0.0	1.15e-04	-	7.68e-02	1.65e-02
0.50	32	8.59e-16	0.0	1.25e-03	2.05	1.67e-02	3.46e-03
	64	4.42e-15	0.0	3.00e-04	2.03	3.51e-02	7.71e-03
	128	1.79e-14	0.0	7.35e-05	-	7.92e-02	1.64e-02
1.00	32	8.59e-16	0.0	9.84e-04	2.06	1.84e-02	3.52e-03
	64	4.54e-15	0.0	2.36e-04	2.07	3.83e-02	7.73e-03
	128	1.79e-14	0.0	5.62e-05	-	8.53e-02	1.64e-02

Table 7: Results of the  $2D$  rotation test case with the Swartz reconstruction [37] with 10 normal correction steps, Taylor integration for the cell displacements, **IDW** point interpolation, **TBDS** flux integration, and pyramid flux volume correction. Times are reported for the execution on a single process on the I0 architecture.

		Ev	Eb	Eg	O(Eg)	Te	Tr
CFL	N						
0.25	32	1.23e-15	0.0	1.75e-03	1.90	1.93e-02	6.96e-03
	64	4.79e-15	0.0	4.68e-04	2.01	4.14e-02	1.53e-02
	128	1.78e-14	0.0	1.16e-04	-	9.30e-02	3.36e-02
0.50	32	9.82e-16	0.0	9.46e-04	1.83	1.99e-02	6.97e-03
	64	4.91e-15	0.0	2.67e-04	1.84	4.20e-02	1.50e-02
	128	1.80e-14	0.0	7.46e-05	-	9.56e-02	3.36e-02
1.00	32	8.59e-16	0.0	7.52e-04	1.80	2.16e-02	6.98e-03
	64	4.79e-15	0.0	2.16e-04	1.94	4.60e-02	1.54e-02
	128	1.78e-14	0.0	5.63e-05	-	0.10	3.43e-02

Table 8: Results of the  $2D$  rotation test case with the **CCNR** reconstruction with 3 reconstruction steps, Taylor integration for the cell displacements, **IDW** point interpolation, **TBDS** flux integration, and pyramid flux volume correction. Times are reported for the execution on a single process on the I0 architecture.



		Ev	Eb	Eg	O(Eg)	Te	Tr
CFL	N						
0.25	32	1.10e-15	0.0	1.68e-03	1.98	1.56e-02	4.40e-03
	64	3.68e-15	0.0	4.25e-04	1.93	3.36e-02	9.52e-03
	128	1.93e-14	0.0	1.11e-04	-	7.86e-02	2.15e-02
0.50	32	9.82e-16	0.0	9.02e-04	2.18	1.64e-02	4.46e-03
	64	3.31e-15	0.0	1.99e-04	1.80	3.51e-02	9.68e-03
	128	1.55e-14	0.0	5.74e-05	-	8.07e-02	2.13e-02
1.00	32	7.37e-16	0.0	1.61e-03	2.00	1.87e-02	4.66e-03
	64	3.81e-15	0.0	4.01e-04	1.88	3.86e-02	9.90e-03
	128	1.42e-14	0.0	1.09e-04	-	8.59e-02	2.12e-02

Table 9: Results of the 2D rotation test case with the **DGNR** reconstruction with 2 reconstruction steps, Taylor integration for the cell displacements, **IDW** point interpolation, **TBDS** flux integration, and pyramid flux volume correction. Times are reported for the execution on a single process on the I0 architecture.

with the **PCFSC-CVTNA** method for all *CFL* numbers, they are comparable to the results of López et al. [75] for *CFL* = 0.5 for their **EMFPA-SIR** method that uses spline-based interface reconstruction applicable only to 2D structured cartesian meshes. The results are also better than those of López et al. [75] with *CFL* = 0.5 and a combination **EMFPA-LVIRA**.

The results with the **DGNR** reconstruction algorithm are shown in table 9. As for **CCNR**, 3 reconstruction steps are used to ensure second-order accuracy. The **DGNR** is based on numerical approximation of the interface normal and it is therefore of somewhat lower accuracy and convergence order compared to **CCNR**.

It is important to note that the interface does not stretch in this test case, which makes the rotation case especially useful for evaluating the computational efficiency. There are two important computational aspects that must be considered when a new method for interface advection is developed: *serial efficiency* and *parallel speedup*.

Serial efficiency is defined as the ratio of the measured computational time and the best possible computational time obtained by an algorithm on the same computing architecture when the program is executed as a single process with no concurrent (parallel) execution of any parts of the algorithm. The best possible execution time is very difficult to predict in the case of the geometrical **VoF** method because of its complexity: it is impossible to theoretically define a single execution unit-step and discuss the methods complexity as  $\mathcal{O}(f(n))$  where  $n$  is the number of said execution unit-steps. All computations of the geometrical **VoF** method, from the transport level down to the geometrical intersections, contain tests that avoid unnecessary computation: tests whose end result depends on the geometrical information of the interface as well as volume fraction field values. As both change abruptly in time and space, analyzing the computational complexity of an optimized geometrical **VoF** method is not possible for a general initial value problem. Because of this, timing measurements taken during the calculation are used to compute a

ratios between the measured execution time for different reconstruction algorithms and the results are summarized in table 10. The same approach to investigating serial efficiency is taken by other authors, as already described in section 5.1. The timing measurement ratios for the overall computational time  $T_e$  in table 10 indicate that the Swartz reconstruction algorithm outperforms both the CCNR and the DGNR reconstruction algorithm. Reconstruction time  $T_r$  ratios show how important it is to consider both the reconstruction time as well as the overall execution time: the reconstruction times of the CCNR and DGNR methods are 4 – 5 times longer than the reconstruction time of the Swartz algorithm in serial, however the overall increase in the execution time  $T_e$  is approximately only 20%.

The parallel implementation of both the hybrid Level Set / Front Tracking method and the geometrical VoF method is based on the domain decomposition and message passing. The reason behind this decision is the fact that the size of the problems that require two-phase DNS methods require the use of distributed memory HPC systems. Parallel speedup is determined using the Amdahl's law [100] modified by Gustafson and Barsis [50]

$$S = \frac{Ts + N Tp}{Ts + Tp + N To} \quad (213)$$

where  $Ts$  and  $Tp$  are the times required for the execution of respective serial and parallel parts of the algorithm and  $N$  is the number of parallel processes. It is important to note that the parallel computation time  $Tp$  consists not only of algorithmic computation, but also of a so-called *parallel overhead*  $To$ : time required to set up the parallel communication and exchange messages required for successful completion of the parallel task. Equation (213) shows that an increase in  $To$  lowers the possible parallel speedup of the algorithm, even more so with the increase in the number of parallel processes.

The results in tables 6 to 9 are computed with the pyramid correction of the geometrical flux. It is often claimed that the iterative approach to correcting the geometrical flux volume for volume conservation introduces prohibitive efficiency issues as well as large errors in volume conservation and numerical stability [26, 65, 98]. Table 11 shows the results of the geometrical VoF method with an iterative scaled geometrical flux volume and the DGNR reconstruction method. Compared to table 9, the results have a maximal 20% increase in computational time. There is no visible influence of the scaled iterative correction on the volume conservation and numerical stability. The order of convergence or absolute accuracy are only slightly lower.

The main conclusion from the results of the two-dimensional rotation is that even though algorithms like Swartz-Mosso, Swartz, LVIRA and ELVIRA may show better serial efficiency, their calculations involve many repetitions when accessing neighboring cell values, which makes them poorly scalable and thus inapplicable to large problems. Proposed new CCNR and DGNR algorithms are second-order accurate, scalable alternatives with direct support for both unstructured and structured Cartesian meshes. Furthermore, a measure of the reconstruction time reported in the literature (see table 1) should not be used without the overall execution time

CFL	0.25			0.50			1.00		
N	32	64	128	32	64	128	32	64	128
Eg S-Y	1.02	0.91	0.51	1.24	0.82	0.41	1.19	0.86	0.45
Te S-Y	1.04	1.10	1.08	1.05	1.09	1.07	1.08	1.07	1.04
Tr S-Y	2.64	2.52	2.29	2.44	2.52	2.28	3.08	2.43	2.18
Eg CCNR-Y	0.96	0.91	0.51	0.94	0.72	0.42	0.91	0.79	0.45
Te CCNR-Y	1.28	1.35	1.31	1.25	1.30	1.29	1.27	1.29	1.26
Tr CCNR-Y	5.42	4.98	4.67	4.92	4.90	4.67	6.11	4.84	4.57
Eg DGNR-Y	1.00	0.91	0.58	1.10	0.80	0.49	1.07	0.87	0.53
Te DGNR-Y	1.28	1.32	1.29	1.24	1.31	1.32	1.29	1.29	1.21
Tr DGNR-Y	5.43	4.83	4.48	4.90	4.88	4.49	6.20	4.74	4.30

Table 10: Algorithm serial execution time and error ratios for the 2D rotation case. Ratios are computed with respect to the Youngs' reconstruction method: Eg S-Y is the ratio of the Eg error between the Swartz and the Youngs reconstruction algorithm. Reconstruction algorithms are used with the same combination of the cell displacement, point-cell interpolation, flux integration and flux correction methods as in tables 6 to 9.

		Ev	Eb	Eg	O(Eg)	Te	Tr
CFL	N						
0.25	32	7.37e-16	0.0	1.81e-03	1.95	2.55e-02	7.17e-03
	64	4.79e-15	0.0	4.69e-04	1.81	5.22e-02	1.54e-02
	128	1.82e-14	0.0	1.34e-04	-	0.11	3.28e-02
0.50	32	7.37e-16	0.0	1.11e-03	1.92	2.70e-02	7.13e-03
	64	4.54e-15	0.0	2.95e-04	1.75	5.44e-02	1.51e-02
	128	1.78e-14	0.0	8.75e-05	-	0.12	3.26e-02
1.00	32	7.37e-16	0.0	8.90e-04	1.90	3.02e-02	7.31e-03
	64	4.42e-15	0.0	2.39e-04	1.76	5.90e-02	1.50e-02
	128	1.77e-14	0.0	7.02e-05	-	0.12	3.21e-02

Table 11: Results of the 2D rotation test case with the DGNR reconstruction with 2 reconstruction steps, Taylor integration for the cell displacements, IDW point interpolation, TBDS flux integration, and scaled flux volume correction. Times are reported for the execution on a single process on the I0 architecture.

To conclude whether the reconstruction algorithm is worth implementing or not from the standpoint of its computational efficiency.

### 11.3 SHEAR

The 2D shear verification case was introduced by R. J. Leveque [109]. The test consists of a circular interface of radius  $R_{s2D} = 0.15$ , centered at  $c_{s2D} = (0.5, 0.75, 0)$ . The interface is evolved using an explicitly prescribed velocity  $\mathbf{u}(u_x(t), u_y(t), 0)$

$$u_x = \sin(2\pi y) \sin^2(\pi x) \cos\left(\frac{\pi t}{T}\right), \quad (214)$$

$$u_y = -\sin(2\pi x) \sin^2(\pi y) \cos\left(\frac{\pi t}{T}\right), \quad (215)$$

with  $CFL = 1$ , in a square unit-length solution domain  $1 \times 1$ . The  $\cos$  multiplier ensures the velocity field switches orientation so that the interface motion reverses and the interface is brought to its initial configuration. Comminal et al. [26] have recently done a tabular comparison of different methods for this test case, and it is shown in table 12 for different mesh densities and periods  $T$  with  $CFL = 1$ . The 2D shear verification case relies on a temporally and spatially varying velocity field and it is a challenging case often used to verify an interface advection method because for a subset of mesh faces the velocities can be almost coplanar as described in section 6.3.3. For these reasons, this case is used to distinguish the effects of different sub-algorithms of the proposed UFVFC scheme.

First, the UFVFC scheme is tested with the Youngs' reconstruction algorithm but with different discrete gradient operators used to estimate the interface normal. Both tables 14 and 15 rely on the IDW velocity interpolation, second-order accurate trapezoidal quadrature for the flux volume integration and the pyramid flux volume correction. However, table 14 shows the results with the Youngs' algorithm that uses the proposed IDW Gauss gradient for normal estimation, while table 15 shows the results with the Least Squares (LS) gradient. As expected, for those cases where the interface reconstruction has the most impact, the order of convergence is 1, with a negligible difference between the IDW Gauss and the LS gradient.

Table 16 shows the increase in accuracy resulting from the proposed modified Youngs' / Swartz Reconstruction algorithm (YSR). The YSR algorithm delivers the most stable second-order convergence and the best absolute accuracy. Results in table 16 extracted into table 13 for comparison are less accurate than those of Comminal et al. [26] shown for the CCU, because Comminal et al. [26] have relied on a more accurate point interpolation and temporal integration scheme. From the Eulerian flux-based methods, OD method proposed by Owkes and Desjardins [98] generates the most accurate results for the 2D shear test case. The Eulerian flux-based method of Jofre et al. [65] is very similar and is expected to provide very similar results, however Jofre et al. [65] have only shown 3D results. Table 13 shows that the UFVFC scheme is more accurate than the OD method and less accurate than the CCU method.

		Eg (32)	Eg (64)	O (32)	Eg (128)	O (64)	Eg (256)	O (128)
T	Method							
0.5	RK	7.29e-04	1.42e-04	2.36	3.90e-05	1.86	—	—
	Stream	5.51e-04	1.10e-04	2.32	3.38e-05	1.71	—	—
	EMFPA	4.45e-04	7.99e-05	2.48	2.04e-05	1.97	—	—
	MZ	4.68e-04	6.91e-05	2.76	2.07e-05	1.74	—	—
	GPCA	4.12e-04	7.32e-05	2.41	1.93e-05	1.93	—	—
	CCU	3.20e-04	7.68e-05	2.06	1.32e-05	2.54	2.45e-06	2.43
2.0	RK	2.36e-03	5.85e-04	2.01	1.31e-04	2.16	—	—
	Stream	2.37e-03	5.65e-04	2.07	1.32e-04	2.10	—	—
	EMFPA	2.14e-03	5.39e-04	1.99	1.29e-04	2.06	—	—
	MZ	2.11e-03	5.28e-04	2.00	1.28e-04	2.05	—	—
	GPCA	2.18e-03	5.32e-04	2.05	1.29e-04	2.03	—	—
	CCU	1.86e-03	4.18e-04	2.15	9.62e-05	2.12	1.97e-05	2.29
8.0	RK	4.78e-02	6.96e-03	2.78	1.44e-03	2.27	—	—
	Stream	3.72e-02	6.79e-03	2.45	1.18e-03	2.52	—	—
	EMFPA	3.77e-02	6.58e-03	2.52	1.07e-03	2.62	2.35e-04	2.19
	MZ	5.42e-02	7.85e-03	2.79	1.05e-03	2.90	—	—
	GPCA	—	—	—	1.17e-03	—	—	—
	OD	—	7.58e-03	—	1.88e-03	2.01	4.04e-04	2.22
	CCU	3.81e-02	4.58e-03	3.06	1.00e-03	2.20	1.78e-04	2.59

Table 12: Results of the 2D shear flow test case on equidistant Cartesian meshes summarized by Comminal et al. [26], presented here for comparison.

		Eg (32)	Eg (64)	O (32)	Eg (128)	O (64)	Eg (256)	O (128)
T	Method							
0.5	OD	1.58e-03	4.43e-04	1.83	1.19e-04	1.89	—	—
	UFVFC	6.75e-04	1.10e-04	2.61	2.35e-05	2.22	6.06e-06	1.95
	CCU	3.20e-04	7.68e-05	2.06	1.32e-05	2.54	2.45e-06	2.43
2.0	OD	2.00e-02	3.33e-03	2.58	8.90e-04	1.9	—	—
	UFVFC	2.12e-03	4.09e-04	2.38	8.88e-05	2.2	2.99e-05	1.57
	CCU	1.86e-03	4.18e-04	2.15	9.62e-05	2.12	1.97e-05	2.29
8.0	OD	—	7.58e-03	—	1.88e-03	2.01	4.04e-04	2.22
	UFVFC	4.58e-02	6.42e-03	2.83	1.41e-03	2.18	3.30e-04	2.10
	CCU	3.81e-02	4.58e-03	3.06	1.00e-03	2.20	1.78e-04	2.59

Table 13: Comparison of the geometrical error of the UFVFC scheme configuration in table 16 with the CCU method of Comminal et al. [26] and OD method of Owkes and Desjardins [98]. Values for  $T = 0.5, 2$  for the OD method are taken from [98, table 3].

The three-dimensional shear verification case has been introduced by Liovic et al. [73]. It consists of a sphere with radius  $R_{s3D} = 0.15$ , centered at  $\mathbf{c}_{s3D} =$

		Ev	Eb	Eg	O(Eg)	En	O(En)	Te	Tr
T	N								
0.5	32	6.14e-16	0.0	3.36e-04	1.00	4.75e-03	1.00	1.58e-02	1.56e-03
	64	2.82e-15	0.0	1.68e-04	0.90	2.38e-03	0.90	3.41e-02	4.03e-03
	128	8.22e-15	0.0	9.02e-05	0.90	1.28e-03	0.90	7.81e-02	9.64e-03
	256	2.43e-14	0.0	4.85e-05	-	6.86e-04	-	0.19	2.48e-02
2.0	32	6.14e-16	0.0	2.53e-03	1.86	3.58e-02	1.86	1.98e-02	2.11e-03
	64	4.54e-15	0.0	7.00e-04	1.70	9.90e-03	1.70	4.29e-02	4.94e-03
	128	1.40e-14	0.0	2.15e-04	1.13	3.04e-03	1.13	9.73e-02	1.20e-02
	256	4.91e-14	0.0	9.79e-05	-	1.39e-03	-	0.23	2.91e-02
8.0	32	9.82e-16	0.0	3.93e-02	2.01	0.56	2.01	4.47e-02	5.37e-03
	64	5.28e-15	0.0	9.78e-03	2.21	0.14	2.21	0.11	1.39e-02
	128	1.61e-14	0.0	2.12e-03	2.01	3.00e-02	2.01	0.22	3.13e-02
	256	5.50e-14	0.0	5.26e-04	-	7.44e-03	-	0.48	6.63e-02

Table 14: Results of the 2D shear test case with the Youngs’ reconstruction trapezoidal point displacement integration with the **IDW** Gauss gradient for the normal orientation, **IDW** point velocity interpolation, trapezoidal flux integration, and pyramid flux volume correction. Times are reported for the execution on a single process on the I0 architecture.

$(0.5, 0.75, 0.25)$ , and the velocity field whose two components are identical to the 2D shear case outlined in section 11.3 and the third component is defined as

$$u_z = u_{max} \left(1 - \frac{r}{R}\right)^2 \cos\left(\frac{\pi t}{T}\right), \quad (216)$$

where  $R = 0.5$  and  $r = \sqrt{(x - \mathbf{c}_{s3D,x})^2 + (y - \mathbf{c}_{s3D,y})^2}$  with  $CFL = 0.5$ . For this test case, the **DGMR** algorithm was used to show it achieves second-order error convergence also in 3D. Compared to the results of Liovic et al. [73], results outlined in table 17 show again volume conservation near machine tolerance and exact numerical boundedness. As for the geometrical errors  $Eg$ , they are comparable to the errors reported by Liovic et al. [73, table 5] with their **CVTNA** method on Cartesian structured meshes ( $2.86e-03$ ,  $7.14e-04$ ,  $1.56e-04$ ) and better than those reported by Jofre et al. [65, table 4] on unstructured hexahedral meshes with the widely used **LVIRA** reconstruction algorithm ( $4.08e-03$ ,  $1.46e-03$ ,  $3.53e-04$ ). The execution times reported in table 17 are computed with four processes on the I1 architecture. The total execution times  $Te$  require additional careful consideration in terms of the time required by the advection algorithm. The focus on the advection algorithm is based on the reported  $Tr$  reconstruction times, that are significantly shorter than the total execution times, making them uninteresting for **HPC** optimization. The parallel implementation of the **UFVFC** scheme is based on the trivial outflow scalar phase flux volume exchange: a simple point-to-point message exchange that is also used by the **DGMR** reconstruction algorithm. Therefore, the communication overhead is most likely not the cause for the increased computational

		Ev	Eb	Eg	O(Eg)	En	O(En)	Te	Tr
T	N								
0.5	32	2.45e-16	0.0	3.36e-04	1.00	4.75e-03	1.00	1.52e-02	1.31e-03
	64	2.58e-15	0.0	1.68e-04	0.88	2.38e-03	0.88	3.29e-02	3.31e-03
	128	6.99e-15	0.0	9.15e-05	0.92	1.29e-03	0.92	7.47e-02	7.47e-03
	256	2.42e-14	0.0	4.85e-05	-	6.86e-04	-	0.18	1.73e-02
2.0	32	8.59e-16	0.0	2.53e-03	1.86	3.58e-02	1.86	1.96e-02	1.91e-03
	64	4.05e-15	0.0	7.00e-04	1.70	9.90e-03	1.70	4.25e-02	4.41e-03
	128	1.31e-14	0.0	2.15e-04	1.15	3.04e-03	1.15	9.18e-02	9.47e-03
	256	5.07e-14	0.0	9.66e-05	-	1.37e-03	-	0.22	2.16e-02
8.0	32	1.60e-15	0.0	3.93e-02	2.01	0.56	2.01	4.39e-02	5.09e-03
	64	5.52e-15	0.0	9.78e-03	2.21	0.14	2.21	0.10	1.31e-02
	128	1.72e-14	0.0	2.12e-03	2.02	3.00e-02	2.02	0.24	3.05e-02
	256	5.69e-14	0.0	5.22e-04	-	7.39e-03	-	0.47	5.85e-02

Table 15: Results of the 2D shear test case with the Youngs' reconstruction trapezoidal point displacement integration with the **LS** Gauss gradient for the normal orientation, **IDW** point velocity interpolation, trapezoidal flux integration, and pyramid flux volume correction. Times are reported for the execution on a single process on the I0 architecture.

time per time step. Instead, the serial scaling of the **UFVFC** scheme must be investigated with the increase of interface elements, as well as the overhead caused by the process load imbalance characteristical for the shear flow verification cases.

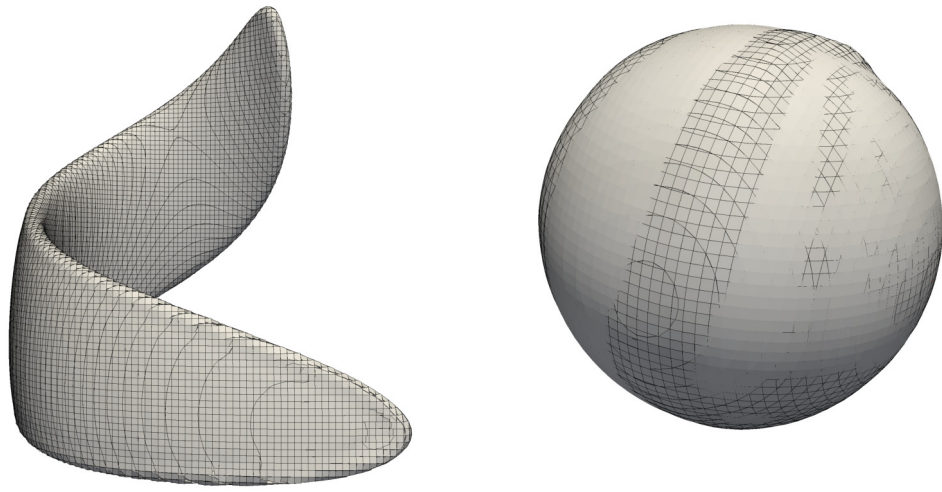
T	N	Ev	Eb	Eg	O(Eg)	En	O(En)	Te	Tr
0.5	32	7.36e-16	0.0	6.75e-04	2.61	9.55e-03	2.61	1.78e-02	4.13e-03
	64	2.33e-15	0.0	1.10e-04	2.23	1.56e-03	2.23	3.96e-02	1.03e-02
	128	7.36e-15	0.0	2.35e-05	1.95	3.32e-04	1.95	9.46e-02	2.72e-02
	256	2.54e-14	0.0	6.06e-06	-	8.57e-05	-	0.24	7.63e-02
2.0	32	8.59e-16	0.0	2.12e-03	2.38	3.01e-02	2.38	2.41e-02	5.84e-03
	64	4.42e-15	0.0	4.09e-04	2.20	5.78e-03	2.20	5.09e-02	1.28e-02
	128	1.36e-14	0.0	8.88e-05	1.57	1.26e-03	1.57	0.12	3.25e-02
	256	5.10e-14	0.0	2.99e-05	-	4.22e-04	-	0.28	8.68e-02
8.0	32	1.47e-15	0.0	4.58e-02	2.83	0.65	2.83	5.27e-02	1.28e-02
	64	5.52e-15	0.0	6.42e-03	2.18	9.08e-02	2.18	0.12	3.19e-02
	128	1.64e-14	0.0	1.41e-03	2.10	2.00e-02	2.10	0.27	7.40e-02
	256	5.62e-14	0.0	3.30e-04	-	4.67e-03	-	0.60	0.18

Table 16: Results of the 2D shear test case with the Swartz reconstruction with 10 normal correction iterations, trapezoidal integration for the cell displacements, **IDW** point interpolation, trapezoidal flux integration, and pyramid flux volume correction. Times are reported for the execution on a single process on the I0 architecture.

CFL	N	Ev	Eb	Eg	O(Eg)	Te	Tr
0.5	32	1.84e-15	0.0	3.74e-03	1.92	0.69	0.14
	64	5.40e-15	0.0	9.91e-04	2.09	3.29	0.61
	128	1.51e-14	0.0	2.32e-04	-	14.57	2.54

Table 17: Results of the 3D shear test case with the **DG NR** reconstruction with 2 normal reconstructions, trapezoidal integration for the cell displacements, **IDW** point interpolation, trapezoidal flux integration, and pyramid flux volume correction. Times are reported for the execution on a single process on the I1 architecture.





(a) 3D shear verification case of Liovic et al. [73] with  $CFL = 0.5$  at  $T = 1.5s$ . (b) 3D shear verification case of Liovic et al. [73] with  $CFL = 0.5$  at  $T = 3s$  (wireframe) and  $T = 0s$  (solid color).

Figure 47: 3D shear verification case of Liovic et al. [73] with  $CFL = 0.5$  and  $N = 128$ , using the new **DG NR** reconstruction algorithm with the parameters that correspond to table 17. Polygons that build the **PLIC** interface are visualized with  $\epsilon_r = 1e - 09$ : no artificial interface separation or wisps are visible.



---

LEVEL SET / FRONT TRACKING METHOD

---

## 12.1 TRANSLATION

The 2D translation verification case of the **LENT** method differs from the one used for the **UFVFC** scheme of the geometrical **VoF** method, covered in section 11.1. The geometrical **VoF** method relies on an interface reconstruction that is performed once per time step. The translation test of the geometrical **VoF** method proposed by Harvie and Fletcher [53] is used to separate the influence of the reconstruction and the advection error and focus on the error caused only by the reconstruction of the interface. Since the interface reconstruction of the **LENT** method can be driven by the curvature estimate, it will not be active for a simple translation. Therefore, there will be no influence of the interface reconstruction. Precisely because the **LENT** does not require an interface reconstruction, the translation test case can be used to verify both the temporal integration, and the marker field model. With the constant velocity field, there is no error in interpolating the velocity for the front vertices, so the interpolation error has no effect on volume conservation.

It is very important to note that one must be very careful not to misinterpret artificial error cancellation that can happen when numerically integrating harmonic functions (cf. Weideman [146]). The temporal truncation error of the Taylor point displacement integration rule defined by equation (167) is  $\mathcal{O}(\delta t^3)$ . Excluding the interpolation error, reconstruction error and the  $\mathcal{O}(\|\mathbf{d}\|^2)$  **FV** discretization error for the  $\nabla_c \mathbf{u}$  discrete differential operator in equation (167) for the simple harmonic translation, it can be expected that the total geometrical error equation (210) is determined by the point displacement integration error  $\mathcal{O}(\delta t^3)$ . The volume conservation should be exact, however this cannot be the case for the **LENT** method as long as an approximative marker field model is used: the **LENT** method relies on the marker field model proposed by Detrixhe and Aslam [31] implemented by Tolle et al. [136]. Therefore, the volume conservation is governed in the translation case by the marker field model. Because the *CFL* condition is not directly applicable to the Lagrangian motion of the fluid interface in the **LENT** method, very high *CFL* numbers can be chosen: the limit is the point-in-mesh search operation implemented by algorithms covered in chapter 9, section 9.1.1.

The 2D translation verification test of the **LENT** method uses a circular interface of radius  $R_{t2D} = 0.15$ , centered at  $\mathbf{c}_{t2D} = (0.25, 0.25, 0)$ . The circle is translated with a harmonic velocity  $\mathbf{u}_{t2D} = (1, 1, 0) \cos(\frac{\pi t}{T})$ , with  $T = 1$  and *CFL* = 3. For this configuration, the first-order accurate *Euler* (rectangle) quadrature rule was used to generate the results shown in table 18, while the third-order accurate

Taylor displacement integration was used for table 19. Table 19 confirms the third-order of the truncation error in the Taylor displacement integration given by equation (167), at the same cost, when the execution time per time step  $Te$  is compared for both temporal integration methods. Very similar execution times are achieved because the computational bottleneck of the LENT method lies in the mesh search operations. Volume conservation errors confirm the second-order accuracy of the implemented marker field (volume fraction) model by Detrixhe and Aslam [31]. To compare visually the difference in the motion of a circle for the coarsest mesh of  $32^2$  volumes, the front is shown at different simulation times for both the Taylor and the Euler displacement integration rule in figure 48. With  $CFL = 3$ , very large time steps can be used, resulting in a visible discrepancy in the positions of the fronts integrated with respective Euler and Taylor displacement integrator. From this and the results presented in the tables it is obvious that the Taylor displacement integration not only converges better, but has absolutely higher accuracy.

		Ev	Eb	Eg	O(Eg)	Te
CFL	N					
3	32	2.91e-04	0.0	8.08e-02	1.08	5.00e-03
	64	7.32e-05	0.0	3.82e-02	0.92	1.55e-02
	128	1.81e-05	0.0	2.02e-02	1.03	4.74e-02
	256	4.55e-06	0.0	9.86e-03	-	0.17

Table 18: Results of the LENT 2D translation test case using the explicit first-order accurate Euler temporal integration and IDW point velocity interpolation.

		Ev	Eb	Eg	O(Eg)	Te
CFL	N					
3	32	2.94e-04	0.0	4.13e-03	3.39	5.30e-03
	64	7.33e-05	0.0	3.93e-04	2.80	1.38e-02
	128	1.79e-05	0.0	5.65e-05	3.10	4.63e-02
	256	4.55e-06	0.0	6.58e-06	-	0.16

Table 19: Results of the LENT 2D translation test case using the explicit third-order accurate Taylor temporal integration and IDW point velocity interpolation.

## 12.2 ROTATION

Compared to the translation verification case, the rotation test of the LENT method adds a spatial gradient to the velocity field  $\mathbf{u}(\mathbf{x}, t)$ . Here the temporal variation of the velocity field is not used, so the main error is the spatial truncation error of the FV discrete gradient operator  $\mathcal{O}(\|\mathbf{d}\|^2)$ , which is reflected in the results.

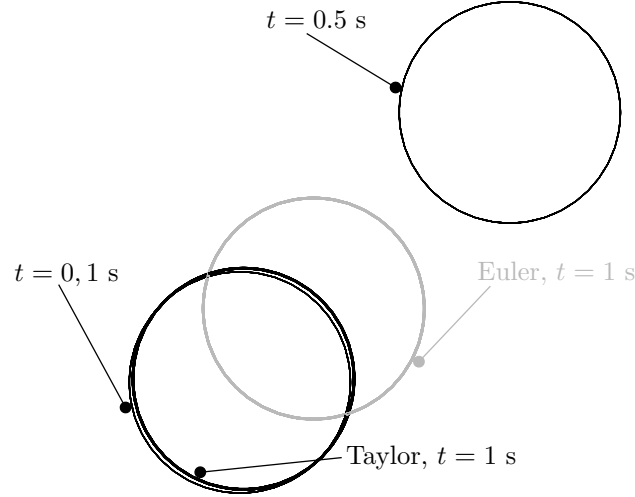


Figure 48: Visual comparison of the temporal integration error of the Euler and Taylor displacement integration rule, isolated for the translation verification case for  $N = 32, T = 1, CFL = 3.0$ .

In 2D, the rotation verification case of the **LENT** method consists of a circular interface of radius  $R_{r2D} = 0.15$ , centered at  $c_{r2D} = (0.5, 0.75, 0)$  and revolved around the axis  $a_{r2D} = (0.5, 0.5, 0)$  for a single complete revolution with the angular velocity  $\omega_{r2D} = (0, 0, 1)$  and  $CFL = 0.25, 0.5, 1$  in a domain of dimensions  $1 \times 1$ .

The **LENT** method maintains a stable second-order convergence for the rotation verification case and delivers absolutely more accurate results in terms of the geometrical error  $Eg$  than the **UFVFC** scheme: from table 7, the **UFVFC**  $Eg$  values with  $CFL = 1$  are  $9.84e-04, 2.36e-04, 5.62e-05$  for  $N = 32, 64, 128$ , respectively. Of course, the **UFVFC** scheme is volume conservative near to machine tolerance, however the **LENT** method is not limited to  $CFL = 1$ , so larger time steps can be used to reduce the total number of calculations.

### 12.3 SHEAR

In the shear flow verification case, the velocity field is varied spatially and temporally. The interface is stretched until thin fillaments develop, that test the overall convergence of the interface advection scheme. The hybrid Level Set / Front Tracking method shows an overall second-order convergent solution in time and space as well as a very good volume conservation property for a Front Tracking scheme, comparable to the Local Front Reconstruction Method (**LFRM**) and Level Contour Reconstruction Method (**LCRM**) of Shin and Juric [128]. The volume conservation is limited by three sub-algorithms in this verification case: interface iso-surface reconstruction algorithm, Lagrange advection of the interface and the marker field (volume fraction) model. Used interface reconstruction algorithm by Treece et al. [137] requires improvements in terms of order of accuracy when the mesh edge is not collinear with the interface normal - as emphasized by Shin and Juric [126]. The Lagrange advection relies currently either on the Taylor series interpolation,

		Ev	Eb	Eg	O(Eg)	Te
CFL	N					
3.0	32	8.11e-03	0.0	6.03e-03	2.08	5.03e-03
	64	1.07e-03	0.0	1.42e-03	2.00	1.58e-02
	128	2.68e-04	0.0	3.56e-04	2.06	7.31e-02
	256	7.65e-05	0.0	8.56e-05	-	0.39
2.0	32	4.37e-03	0.0	2.67e-03	2.05	5.35e-03
	64	1.12e-03	0.0	6.45e-04	2.06	1.59e-02
	128	2.99e-04	0.0	1.54e-04	1.91	7.25e-02
	256	7.59e-05	0.0	4.10e-05	-	0.38
1.0	32	4.63e-03	0.0	6.43e-04	2.01	5.16e-03
	64	1.18e-03	0.0	1.60e-04	2.02	1.59e-02
	128	2.97e-04	0.0	3.93e-05	1.94	7.29e-02
	256	7.63e-05	0.0	1.02e-05	-	0.38
0.5	32	4.77e-03	0.0	1.87e-04	2.18	5.22e-03
	64	1.20e-03	0.0	4.12e-05	1.89	1.66e-02
	128	3.30e-04	0.0	1.11e-05	1.84	7.20e-02
	256	8.18e-05	0.0	3.12e-06	-	0.38

Table 20: Results of the **LENT** 2D rotation test case using the explicit Taylor temporal integration and **IDW** point velocity interpolation.

or the **IDW** velocity interpolation; higher-order interpolations, or interpolations with a divergence-free basis might be used to increase volume conservation.

The two-dimensional shear flow verification case setup is outlined in section 11.3. As for the **LCRM** method, a higher-order temporal discretization scheme is just as vital for the **LENT** method. Results presented in table 21 confirm the significant increase of the volume conservation error when a simple Euler displacement integration is used, compared to the Taylor displacement integration shown in table 22. Using the Taylor displacement integration does not cause a significant increase in the total computational time per time step (Te). The Taylor displacement integration leads to higher-order of convergence (significantly above 2) in table 22. Therefore, the hybrid Level Set / Front Tracking method has a potential to have an absolute accuracy and convergence order higher than geometrical **VoF** method, provided that volume conservation is significantly improved. As for the rotation case, the volume conservation error converges with second-order accuracy.

Figures 49 and 50 contain the actual geometry of the front at different times  $t = 0, 0.5T, T$ , confirming visually the results presented in tables 21 and 22 for the mesh with  $256^2$  volumes. Even for an insignificant interface deformation with  $T = 0.5$ , there is a visible difference between the interface at  $T = 0$  and  $T = 0.5$  when Euler displacement integration is used. For Taylor displacement integration, this difference is no longer visible. It is important to note that the results outlined in this section are computed with  $CFL = 1$  for comparison with the geometrical **VoF** method, although higher  $CFL = 3$  can be safely used without incurring additional

		Ev	Eb	Eg	O(Eg)	Te
T	N					
0.5	32	3.46e-03	0.0	1.81e-02	0.98	5.44e-03
	64	3.71e-03	0.0	9.16e-03	1.00	1.84e-02
	128	2.39e-03	0.0	4.58e-03	1.00	8.72e-02
	256	1.32e-03	0.0	2.29e-03	-	0.50
2.0	32	3.50e-02	0.0	4.71e-02	0.97	5.73e-03
	64	2.44e-03	0.0	2.40e-02	1.00	2.02e-02
	128	2.90e-03	0.0	1.20e-02	1.01	9.43e-02
	256	2.47e-03	0.0	5.96e-03	-	0.51
8.0	32	0.50	0.0	0.11	-0.29	8.96e-03
	64	0.16	0.0	0.13	0.21	3.04e-02
	128	3.23e-02	0.0	0.11	0.98	0.13
	256	2.87e-03	0.0	5.72e-02	-	0.58

Table 21: Results of the **LENT** 2D shear test case using the explicit Euler temporal integration and **IDW** point velocity interpolation.

		Ev	Eb	Eg	O(Eg)	Te
T	N					
0.5	32	5.78e-03	0.0	3.02e-04	2.81	5.31e-03
	64	1.40e-03	0.0	4.30e-05	2.46	1.79e-02
	128	3.30e-04	0.0	7.80e-06	2.10	8.69e-02
	256	8.59e-05	0.0	1.82e-06	-	0.49
2.0	32	1.59e-02	0.0	3.93e-04	2.75	5.72e-03
	64	3.00e-03	0.0	5.83e-05	2.55	1.98e-02
	128	7.00e-04	0.0	9.97e-06	2.22	9.41e-02
	256	1.65e-04	0.0	2.15e-06	-	0.50
8.0	32	0.26	0.0	3.05e-03	2.93	8.59e-03
	64	7.83e-02	0.0	4.00e-04	2.95	2.87e-02
	128	9.94e-03	0.0	5.17e-05	2.95	0.12
	256	2.49e-03	0.0	6.67e-06	-	0.58

Table 22: Results of the **LENT** 2D shear test case using the explicit Taylor temporal integration and **IDW** point velocity interpolation.

computational costs by increasing the width to used for the the narrow band signed distance calculation to  $> 3$  cells in the direction of the interface normal.

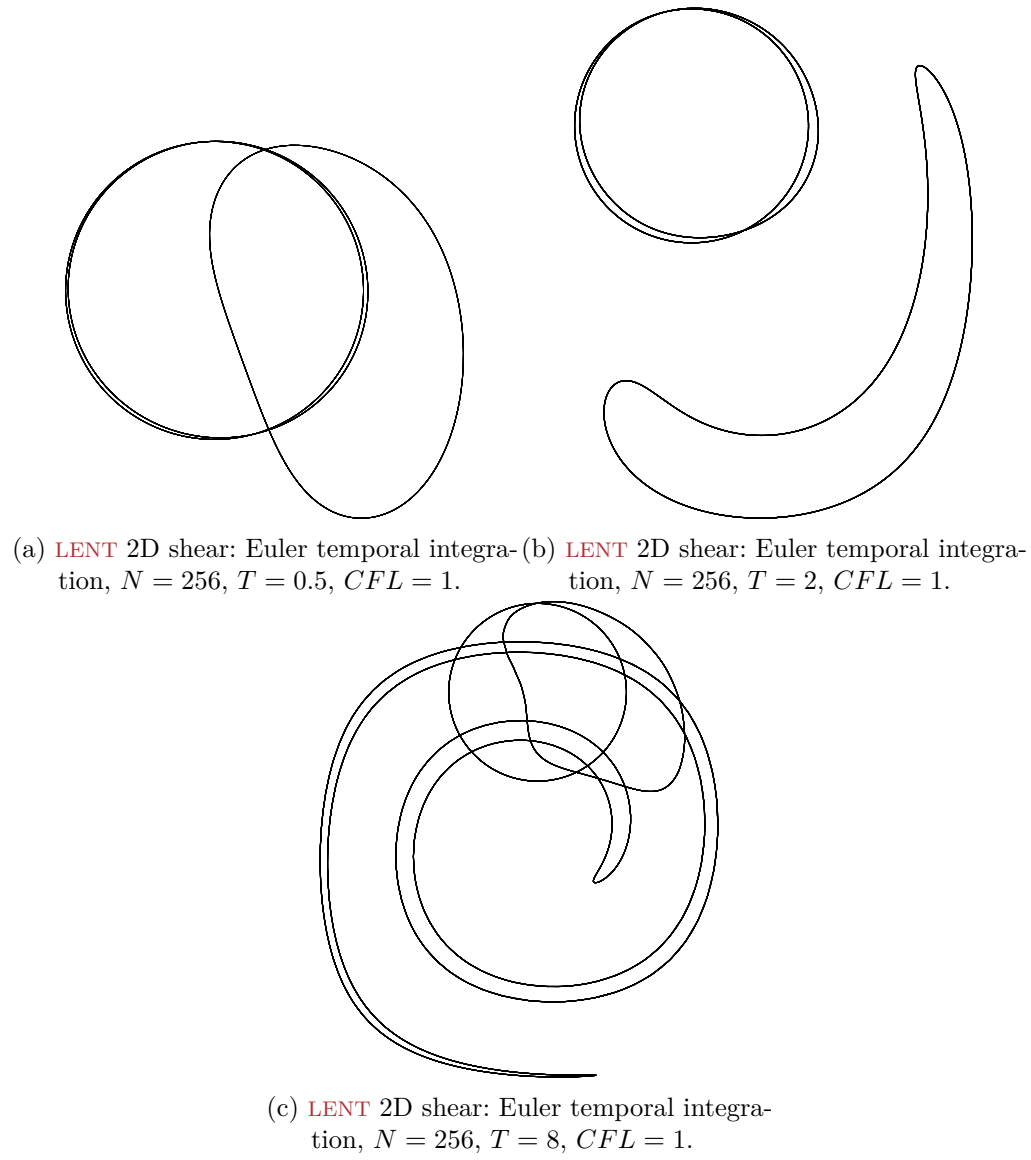


Figure 49: Visualization of the front at the initial, half and final period of motion for the **LENT** two-dimensional shear case with the Euler explicit temporal integration and the **IDW** point-velocity interpolation.



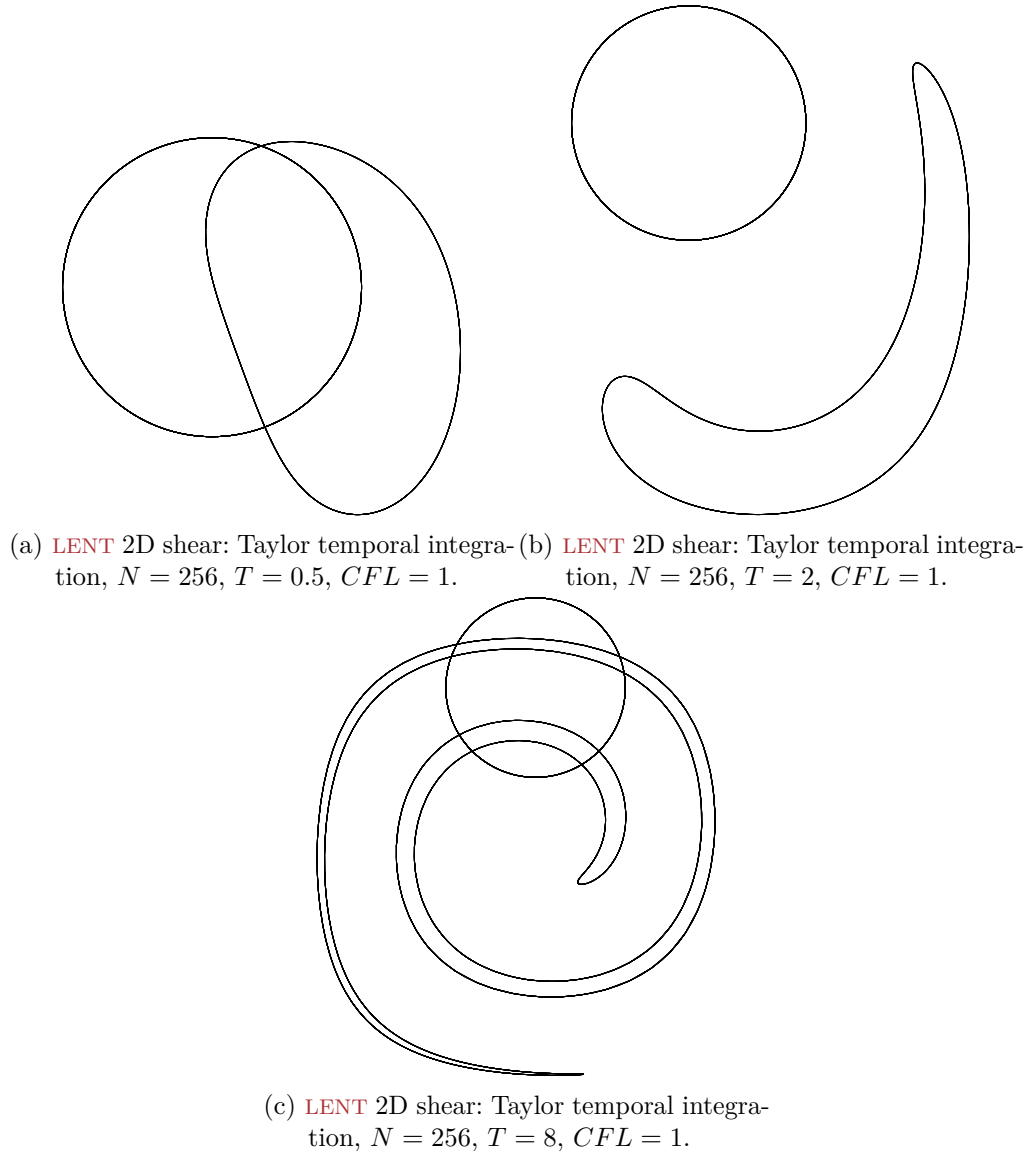


Figure 50: Visualization of the front at the initial, half and final period of motion for the **LENT** two-dimensional shear case with the Taylor explicit temporal integration and the **IDW** point-velocity interpolation.



---

## SUMMARY AND OUTLOOK

---

### 13.1 SUMMARY

A dimensionally un-split geometrical Volume-of-Fluid method on unstructured meshes has been developed and described within this thesis. The verification of the interface advection problem against exact solutions shows a second-order accurate, exactly conservative and exactly numerically bounded transport of the volume fraction field in both time and space. A new triangulation algorithm for congruent polyhedra with non-planar self-intersecting faces increases the accuracy, compared to the oriented and barycentric triangulation of flux polyhedra. Serial execution efficiency is achieved by introducing a reduction of complexity in the flux contribution calculation. An efficient second-order accurate interface reconstruction algorithms for structured and unstructured meshes is proposed: the **DG NR** algorithm. Additionally, the algorithm Swartz (**YSR**) algorithm was slightly modified and combined with the Youngs's algorithm in a way that avoids the increase of artificial numerical surface tension reported by other authors.

A novel coupled Level Set / Front Tracking (**LE NT**) method on unstructured meshes is proposed with an efficient combination of the octree and known-vicinity search algorithms for fast Front Tracking. Higher-order explicit single step point displacement integration method is proposed, together with an efficient second-order cell-to-point interpolation on unstructured meshes. The **LE NT** method shows promising results in term of both high order temporal and second-order spatial error convergence. Its object-oriented design makes it possible to extend it easily in the future.

Both methods are implemented in a modular and computationally efficient way using the C++ programming language. An efficient software library for geometrical operations on arbitrary polyhedra independent of the OpenFOAM platform for CFD was implemented. Its design is based on C++ template metaprogramming, that allows straightforward introduction of new geometrical concepts and extension of the existing set of geometrical algorithms. The geometrical library is used by the transport library layer to assemble new transport algorithms, reducing the time consuming modifications usually encountered during method development.

The methods have been parallelized using the domain decomposition and message passing parallelization method, while sub-algorithms that benefit from it have been developed with support for hybrid domain decomposition / shared memory parallel processing.

### 13.2 OUTLOOK

The initial implementation of both hybrid Level Set / Front Tracking method and geometrical **VoF** method is to be studied in detail and improved further to ensure better serial and parallel scaling and efficiency. As already mentioned, this work is already underway.

As for the methodological improvements, the **PAM** and **CCU** methods of Zhang and Fogelson [153] and Comminal et al. [26] provide higher-order accurate results for the entire family of standardized verification cases. However, they have so far been implemented only in 2D. The self-intersection of the flux polyhedra that cannot be avoided by the Eulerian flux-based methods is less prevalent for the **LE** geometrical **VoF** methods. The correction for volume conservation proposed by Comminal et al. [26] applied to 3D cells using the geometrical framework developed in this thesis could lead to a more robust and accurate geometrical algorithm for the interface advection. A 3D extension of **iPAM**, **CCU**, **MoF** and similar **LE** geometrical **VoF** methods requires accurate geometrical calculations in three dimensions, that are made available in the implemented software framework of the proposed **UFVFC** scheme and **LENT** method, in the form of modular algorithms that can be easily combined into new methods.

In parallel to the further enhancements of the interface evolution, the next step in the development of the proposed methods is the coupling with the single-field two-phase **NS** equation system with the goal of reaching predictive **DNS** of surface tension driven flows in geometrically complex domains. Initial coupling of the hybrid Level Set / Front Tracking method with the **NS** has already shown stable results [78] and further work is still required in order to simulate problems with strong surface tension effects.

**Part V**

**Appendix**



---

## MINIMIZATION OF THE LLSG ERROR

---

Equations introduced in section 5.1.1.1 are repeated here for clarity. Ahn and Shashkov [5] approach the approximation of  $\nabla_c \alpha$  with a linear approximation of the volume fraction field

$$\alpha_l(\mathbf{x}) \approx \alpha_c + \nabla_c \alpha \cdot (\mathbf{x} - \mathbf{x}_c) \quad (217)$$

The volume fraction error for the cell neighborhood  $\mathcal{C}$  is defined by equation (217)

$$\epsilon_l = \sum_{n \in \mathcal{C}} \left( \alpha_n - \frac{\int_{V_n} \alpha_l(\mathbf{x}) dx}{V_n} \right)^2. \quad (218)$$

It represents the difference between the actual volume fraction in the neighbor cell  $\alpha_n$  and the volume fraction in the neighbor cell estimated by equation (217). The integral term in 218

$$\int_{V_n} \alpha_l(\mathbf{x}) dx \quad (219)$$

requires simplification, as follows

$$\begin{aligned} \int_{V_n} \alpha_l(\mathbf{x}) dx &= \int_{V_n} \alpha_c dx + \nabla_c \alpha \cdot (\mathbf{x} - \mathbf{x}_c) dx \\ &= \int_{V_n} \alpha_c dx + \int_{V_n} \nabla_c \alpha \cdot (\mathbf{x} + \mathbf{x}_n - \mathbf{x}_n - \mathbf{x}_c) dx \\ &= \alpha_c V_n + \int_{V_n} \nabla_c \alpha \cdot (\mathbf{x} + \mathbf{x}_n - \mathbf{x}_n - \mathbf{x}_c) dx \\ &= \alpha_c V_n + \int_{V_n} \nabla_c \alpha \cdot (\mathbf{x} - \mathbf{x}_n) dx + \int_{V_n} \nabla_c \alpha \cdot (\mathbf{x}_n - \mathbf{x}_c) dx. \end{aligned} \quad (220)$$

The terms  $\mathbf{x}_n, \mathbf{x}_c$  and  $\nabla_c \alpha$  are constant in  $V_n$ . Additionally,  $\mathbf{x}_n$  is the geometrical center of  $V_n$  and by definition of the geometrical centroid equation (26)

$$\int_{V_n} \nabla_c \alpha \cdot (\mathbf{x} - \mathbf{x}_n) dx = \nabla_c \alpha \cdot \int_{V_n} (\mathbf{x} - \mathbf{x}_n) dx = 0. \quad (221)$$

Above conditions applied to equation (219) result in

$$\int_{V_n} \alpha_l(\mathbf{x}) dx = \alpha_c V_n + \nabla_c \alpha \cdot (\mathbf{x}_n - \mathbf{x}_c) V_n. \quad (222)$$

Inserting equation (222) into equation (218) leads to

$$\min \epsilon_l = \min \sum_{n \in \mathcal{C}} [\alpha_n - \alpha_c - \nabla_c \alpha \cdot (\mathbf{x}_n - \mathbf{x}_c)]^2. \quad (223)$$

The components of the gradient  $\nabla_c \alpha$  in the equation above represent the unknown variables that need to be solved for by the minimization process. The minimization of  $\epsilon_l$  is given by

$$\begin{aligned} \frac{\partial \epsilon_l}{\nabla_{c,x} \alpha} &= 0, \\ \frac{\partial \epsilon_l}{\nabla_{c,y} \alpha} &= 0, \\ \frac{\partial \epsilon_l}{\nabla_{c,z} \alpha} &= 0. \end{aligned} \quad (224)$$

Applying the above equation to equation (223) finally leads to the  $3 \times 3$  linear algebraic system that is solved in each cell  $c$  for the components of the gradient of the  $\alpha$  field:

$$n - 2 [\alpha_n - \alpha_c - \nabla_{c,i} \alpha (i_n - i_c)] (i_n - i_c), \quad (225)$$

where  $i = x, y, z$ . Notice one important thing: the number of linear algebraic equations in the system does not change with the size of  $\mathcal{C}$ . However, the algebraic coefficients do change. Introducing a larger  $\mathcal{C}$  thus results in the increased number of neighboring cells that influence the gradient calculation.



---

INTERFACE MESH GENERATION

---

Interface mesh generation relies on unstructured meshes that can represent the fluid interface accurately by local Adaptive Mesh Refinement (AMR). Two proposed algorithms, the SMCI algorithm 4 and CCI algorithm 3 require as input an unstructured surface mesh and a volume mesh, respectively. Following sections contain notes that are important for the initialization process. Parameters listed in appendices B.1 and B.2 are required to reproduce the results in chapters 11 and 12.

## B.1 INTERFACE SURFACE MESH

Surface meshes used to approximate the initial (exact) interface  $\Gamma_h$  required by the SMCI algorithm 4 are generated in this thesis using the *gmsh* open source mesh generation software. For the results presented in this thesis version 2.12.0 of *gmsh* was used. Source code listing B.1 contains the *gmsh* input data: the most important parameters are `positionZ1`, `positionZ2` and `meshLength`. Since the UFVFC scheme is implemented in pseudo-2D, parameters `positionZ1`(2 determine the height of the cylinder. This height can correspond to the height of the single layer of mesh cells used to discretize the domain  $\Omega_h$ , because the interval-based logic of the implemented geometrical operations will handle special cases of co-planarity. The `meshLength` parameter determines the resolution used to approximate the initial interface  $\Gamma_h$  (cf. section 6.1). In the case of source code listing B.1, the `meshLength` specifies the mesh lengthscale of the triangle surface mesh, for the points that are used to define the geometry of the cylinder.

It is very important to note that the resolution used to approximate the initial interface sets the limit for the absolute accuracy of the numerical method used to approximate the evolution of the interface, the solution given by the UFVFC scheme can only be as accurate as the initial field computed by  $\Gamma_h \cap \Omega_h$  using the SMCI algorithm.

## B.2 INTERFACE VOLUME MESH

The CCI algorithm 3 uses the initial interface  $\Gamma_h$  generated with the input data from appendix B.1, and generates a *volume* mesh used for  $\Gamma_h \cap \Omega_h$  calculation in order to compute  $\alpha_c(t = t_0)$ . For the volume mesh generation, *cfMesh* version v1.1.2 was used. Source code listing B.2 contains the mesh generation parameter for the cylindrical interfaces used by the CCI algorithm. It is important to note that

Listing B.1: Cylinder surface mesh parameter

```

radius=0.15;
meshLength = 0.0008;
positionX = 0.5;
positionY = 0.75;
positionZ1 = 0;
positionZ2 = 0.1;

Point(1) = {positionX, positionY, positionZ1, meshLength};
Point(2) = {positionX-radius, positionY, positionZ1, meshLength};
Point(3) = {positionX, positionY+radius, positionZ1, meshLength};
Point(4) = {positionX+radius, positionY, positionZ1, meshLength};
Point(5) = {positionX, positionY-radius, positionZ1, meshLength};
Point(6) = {positionX, positionY, positionZ2, meshLength};
Point(7) = {positionX-radius, positionY, positionZ2, meshLength};
Point(8) = {positionX, positionY+radius, positionZ2, meshLength};
Point(9) = {positionX+radius, positionY, positionZ2, meshLength};
Point(10) = {positionX, positionY-radius, positionZ2, meshLength};

Circle(1) = {2, 1, 3};
Circle(2) = {3, 1, 4};
Circle(3) = {4, 1, 5};
Circle(4) = {5, 1, 2};
Circle(5) = {7, 6, 8};
Circle(6) = {8, 6, 9};
Circle(7) = {9, 6, 10};
Circle(8) = {10, 6, 7};
Line(9) = {2, 7};
Line(10) = {3, 8};
Line(11) = {4, 9};
Line(12) = {5, 10};
Line Loop(14) = {9, 5, -10, -1};
Ruled Surface(14) = {14};
Line Loop(16) = {10, 6, -11, -2};
Ruled Surface(16) = {16};
Line Loop(18) = {11, 7, -12, -3};
Ruled Surface(18) = {18};
Line Loop(20) = {12, 8, -9, -4};
Ruled Surface(20) = {20};
Surface Loop(26) = {14, 16, 18, 20};

```

the mesh refinement near the interface applied to the volumetric mesh is the same as the one used for the surface mesh in source code listing [B.1](#).

Listing B.2: Cylinder volume mesh parameters

```

surfaceFile      "cylinder.stl";
maxCellSize      0.1;

boundaryLayers
{
    // Local settings applied to patches.
    patchBoundaryLayers
    {
        // Patch name. Supports regular expressions.
        "cylinder"
        {
            // Number of boundary layers at walls.
            nLayers      5;

            // Thickness ratio between consecutive layers.
            thicknessRatio      1.2;
        }
    }
}

localRefinement
{
    // Local refinement regions
    "cylinder"
    {
        cellSize 0.0008;
    }
}

```



---

## BIBLIOGRAPHY

---

- [1] boost.geometry documentation, version 1.55. [http://www.boost.org/doc/libs/1\\_55\\_0/libs/geometry/doc/html/index.html](http://www.boost.org/doc/libs/1_55_0/libs/geometry/doc/html/index.html). Accessed: May 20 2014.
- [2] Generic Programming in C++: Techniques. <http://www.generic-programming.org/languages/cpp/techniques.php>. Accessed: May 20 2014.
- [3] Gilou Agbaglah, Sébastien Delaux, Daniel Fuster, Jérôme Hoepffner, Christophe Josserand, Stéphane Popinet, Pascal Ray, Ruben Scardovelli, and Stéphane Zaleski. Parallel simulation of multiphase flows using octree adaptivity and the volume-of-fluid method. *Comptes Rendus Mécanique*, 339 (2-3):194 – 207, 2011. High Performance Computing.
- [4] Fog Agner. Optimizing software in c++ an optimization guide for windows, linux and mac platforms. Technical report, Technical University of Denmark, 2014.
- [5] H. T. Ahn and M. Shashkov. Multi-material interface reconstruction on generalized polyhedral meshes. *Journal of Computational Physics*, 226(2), 2007.
- [6] Hyung Ahn and Mikhail Shashkov. Geometric Algorithms for 3D Interface Reconstruction. In *Proceedings of the 16th International Meshing Roundtable*, pages 405–422. Springer Berlin Heidelberg, 2008.
- [7] Hyung Taek Ahn and Mikhail Shashkov. Multi-material interface reconstruction on generalized polyhedral meshes. Technical Report LA-UR-07-0656, Los Alamos National Laboratory, 2007. URL [http://cnls.lanl.gov/~shashkov/papers/3D\\_MOF.pdf](http://cnls.lanl.gov/~shashkov/papers/3D_MOF.pdf).
- [8] Hyung Taek Ahn and Mikhail Shashkov. Geometric algorithms for 3D interface reconstruction. In *Proc. 16th Int. Meshing Roundtable, IMR 2007*, pages 405–422, 2008. ISBN 9783540751021. doi: 10.1007/978-3-540-75103-8\_23.
- [9] Hyung Taek Ahn and Mikhail Shashkov. Adaptive moment-of-fluid method. *J. Comput. Phys.*, 228(8):2792–2821, 2009. ISSN 00219991. doi: 10.1016/j.jcp.2008.12.031. URL <http://dx.doi.org/10.1016/j.jcp.2008.12.031>.
- [10] Hyung Taek Ahn and Mikhail Shashkov. Adaptive moment-of-fluid method. *Journal of Computational Physics*, 228(8):2792 – 2821, 2009.

- [11] Hyung Taek Ahn, Mikhail Shashkov, and Mark A. Christon. The moment-of-fluid method in action. *Commun. Numer. Methods Eng.*, 25(10):1009–1018, 2009. ISSN 10698299. doi: 10.1002/cnm.1135. URL <http://onlinelibrary.wiley.com/doi/10.1002/cnm.1494/full>.
- [12] Andrei Alexandrescu. *Modern C++ design: generic programming and design patterns applied*. Addison-Wesley Longman Publishing Co., Inc., 2001. ISBN 0-201-70431-5.
- [13] Kendall E Atkinson. *An introduction to numerical analysis*. John Wiley & Sons, 1989.
- [14] E. Aulisa, S. Manservigi, R. Scardovelli, and S. Zaleski. Interface reconstruction with least-squares fit and split advection in three-dimensional Cartesian geometry. *J. Comput. Phys.*, 225(2):2301–2319, 2007. ISSN 00219991. doi: 10.1016/j.jcp.2007.03.015.
- [15] Eugenio Aulisa, Sandro Manservigi, Ruben Scardovelli, and Stephane Zaleski. A geometrical area-preserving Volume-of-Fluid advection method. *J. Comput. Phys.*, 192(1):355–364, 2003. ISSN 00219991. doi: 10.1016/j.jcp.2003.07.003.
- [16] R. Ernst and B. Stroustrup and A. Sutton, A. Gangolli, J. Kalb, A. Lumsdaine, P. McJones, S. Parent, D. Rose, A. Stepanov, A. Sutton, L. Voufo, J. Willcock, and M. Zalewski. *A Concept Design for the STL*. Technical Report N3351=12-0041, C++ Evolution Working Group, 2012.
- [17] S. Basting and M. Weismann. A hybrid level set–front tracking finite element approach for fluid-structure interaction and two-phase flow applications. *Journal of Computational Physics*, 2013.
- [18] J B Bell, C N Dawson, and G R Shubin. An unsplit, higher-order Godunov method for scalar conservation laws in multiple dimensions. *J. Comput. Phys.*, 74:1–24, 1988.
- [19] J. Bloomenthal. An implicit surface polygonizer. In *Graphics Gems IV*, pages 324–349. Academic Press, 1994.
- [20] R. P. Brent. An algorithm with guaranteed convergence for finding a zero of a function, 1971. ISSN 0010-4620.
- [21] Richard P Brent. *Algorithms for minimization without derivatives*. Dover Publications, 2013.
- [22] H. D. Cenicerros, A. M. Roma, A. Silveira-Neto, and M. M. Villar. A Robust, Fully Adaptive Hybrid Level-Set/Front-Tracking Method for Two-Phase Flows with an Accurate Surface Tension Computation. *Communications in Computational Physics*, 2010.
- [23] Hector D. Cenicerros, Alexandre M. Roma, Aristeu Silveira-Neto, and Milena M. Villar. A Robust, Fully Adaptive Hybrid Level-Set/Front-Tracking

- Method for Two-Phase Flows with an Accurate Surface Tension Computation. *Communications in Computational Physics*, 229:6135 – 6155, 2010.
- [24] Gregor Cerne, Stojan Petelin, and Iztok Tiselj. Numerical errors of the volume-of- fluid interface tracking algorithm. *Int. J. Numer. Methods Fluids*, 38(November 2000):329–350, 2002.
  - [25] Siu-Wing Cheng, Tamal K Dey, and Jonathan Shewchuk. *Delaunay mesh generation*. CRC Press, 2013. ISBN 978-1-58488-731-7.
  - [26] R. Comminal, J. Spangenberg, and J. H. Hattel. Cellwise conservative unsplit advection for the volume of fluid method. *J. Comput. Phys.*, 283:582–608, 2015.
  - [27] C.D. Correa, R. Hero, and Kwan-Liu Ma. A Comparison of Gradient Estimation Methods for Volume Rendering on Unstructured Meshes. *Visualization and Computer Graphics, IEEE Transactions on*, 17:305 –319, 2011. doi: 10.1109/TVCG.2009.105.
  - [28] R DeBar. Fundamentals of the kraken code. *Lawrence Livermore Laboratory, UCIR-760*, 1974.
  - [29] I. Demirdzic and M. Peric. Space conservation law in finite volume calculations of fluid flow. *Int. J. Numer. METHODS FLUIDS*, 8(September 1987):1037–1050, 1988.
  - [30] Suraj S Deshpande, Lakshman Anumolu, and Mario F Trujillo. Evaluating the performance of the two-phase flow solver interFoam. *Computational Science & Discovery*, 5(1):014016, 2012.
  - [31] Miles Detrixhe and Tariq D Aslam. From level set to volume of fluid and back again at second-order accuracy. *Int. J. Numer. METHODS FLUIDS*, 80(August 2015):231–255, 2016. doi: 10.1002/fld.
  - [32] L. Satyan Devadoss and Joseph O’Rourke. *Discrete and Computational Geometry*. Princeton University Press, 2011. ISBN ISBN 978-0-691-14553-2.
  - [33] Kathrin Dieter-Kissling, Holger Marschall, and Dieter Bothe. Direct Numerical Simulation of droplet formation processes under the influence of soluble surfactant mixtures. *Comput. Fluids*, 113:93–105, 2015. ISSN 00457930. doi: 10.1016/j.compfluid.2015.01.017. URL <http://www.sciencedirect.com/science/article/pii/S0045793015000262>.
  - [34] Steven Diot and Marianne M. François. An interface reconstruction method based on an analytical formula for 3D arbitrary convex cells. *J. Comput. Phys.*, 305:63–74, 2016. ISSN 00219991. doi: 10.1016/j.jcp.2015.10.011. URL <http://linkinghub.elsevier.com/retrieve/pii/S0021999115006749>.

- [35] Steven Diot, Marianne M François, and Edward D Dendy. An interface reconstruction method based on analytical formulae for 2d planar and axisymmetric arbitrary convex cells. *Journal of Computational Physics*, 275: 53–64, 2014.
- [36] J. Du, B. Fix, J. Glimm, X. Jia, X. Li, Y. Li, and L. Wu. A simple package for front tracking. *J. Comput. Phys.*, 213(2):613–628, April 2006.
- [37] Vadim Dyadechko and Mikhail Shashkov. Moment-of-fluid interface reconstruction. Technical report, Los Alamos National Laboratory, 2005.
- [38] Vadim Dyadechko and Mikhail Shashkov. Reconstruction of multi-material interfaces from moment data. *Journal of Computational Physics*, 227, May 2008.
- [39] Christer Ericson. *Real-time collision detection*. Morgan Kaufmann, 2005. ISBN 1-55860-732-3.
- [40] Andreas Fabri, Geert-Jan Giezeman, Lutz Kettner, Stefan Schirra, and Sven Schsnherr. The CGAL Kernel : A Basis for Geometric Computation. In Ming C. LinDinesh Manocha, editor, *FCRC’96 Work.*, volume 1, Philadelphia, 1996. doi: DOI:10.1007/BFb0014474.
- [41] Joel H Ferziger and Milovan Perić. *Computational methods for fluid dynamics*, volume 3. Springer Berlin, 2002.
- [42] Agner Fog. *Optimizing software in C++: An optimization guide for Windows, Linux and Mac platforms*, 2015.
- [43] Marianne M. Francois, Sharen J. Cummins, Edward D. Dendy, Douglas B. Kothe, James M. Sicilian, and Matthew W. Williams. A balanced-force algorithm for continuous and sharp interfacial surface tension models within a volume tracking framework. *Journal of Computational Physics*, 213(1):141 – 173, 2006.
- [44] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN 0-201-63361-2.
- [45] Rao Garimella, Milan Kucharik, and Mikhail Shashkov. An efficient linearity and bound preserving conservative interpolation (remapping) on polyhedral meshes. *Computers & Fluids*, 36(2):224 – 237, 2007. ISSN 0045-7930. doi: <http://dx.doi.org/10.1016/j.compfluid.2006.01.014>. URL <http://www.sciencedirect.com/science/article/pii/S0045793006000351>.
- [46] B. Gehrels and B. Lalande. A generic geometry library. In *boostcon / The Boost C++ Libraries Conference*, 2009.
- [47] B. Gehrels, B. Lalande, and Mateusz Loskot. Generic programming for geometry. In *boostcon / The Boost C++ Libraries Conference*, 2010.



- [48] Barend Gehrels, Bruno Lalande, Mateusz Loskot, Adam Wulkiewicz, and Menelaos Karavelas. Boost geometry library manual. [http://www.boost.org/doc/libs/1\\_63\\_0/libs/geometry/doc/html/index.html](http://www.boost.org/doc/libs/1_63_0/libs/geometry/doc/html/index.html). Accessed: 2017-04-04.
- [49] J. Glimm, J. W. Grove, , X. L. Li, K.-M. Shyue, Y. Zeng, and Q. Zhang. Three-dimensional Front Tracking. *SIAM J. Sci. Comput.*, 19:703–727, May 1998.
- [50] John L Gustafson. Reevaluating amdahl’s law. Technical Report 5, 1988.
- [51] Peter Hachenberger, Lutz Kettner, and Kurt Mehlhorn. Boolean operations on 3D selective Nef complexes : Data structure , algorithms , optimized implementation and experiments. *Computational Geometry*, 38:64–99, 2007. doi: 10.1016/j.comgeo.2006.11.009.
- [52] Dalton J. E. Harvie and David F. Fletcher. A new volume of fluid advection algorithm: The defined donating region scheme. *Int. J. Numer. Methods Fluids*, 35(2):151–172, 2001. ISSN 02712091. doi: 10.1002/1097-0363(20010130)35:2<151::AID-FLD87>3.0.CO;2-4.
- [53] Dalton J.E. Harvie and David F. Fletcher. A New Volume of Fluid Advection Algorithm: The Stream Scheme. *J. Comput. Phys.*, 162(1):1–32, 2000. ISSN 00219991. doi: 10.1006/jcph.2000.6510. URL <http://www.sciencedirect.com/science/article/pii/S0021999100965100>.
- [54] J. Hernández, J. López, P. Gomez, C. Zanzi, and F. Faura. A new volume of fluid method in three dimensions—Part I: Multidimensional advection method with face-matched flux polyhedra. *Int. J. Numer. Methods Fluids*, (October 2007):601–629, 2008.
- [55] Charles Hirsch. *Numerical Computation of Internal and External Flows: The Fundamentals of Computational Fluid Dynamics: The Fundamentals of Computational Fluid Dynamics*, volume 1. Butterworth-Heinemann, 2007.
- [56] C. W. Hirt and B. D. Nichols. Volume of fluid (VOF) method for the dynamics of free boundaries. *Journal of computational physics*, 39(1):201–225, 1981.
- [57] J. D. Hoffman and S. Frankel. *Numerical methods for engineers and scientists*. CRC press, 2001.
- [58] J. Hua, J. F. Stene, and P. Lin. Numerical simulation of 3D bubbles rising in viscous liquids using a front tracking method. *Journal of Computational Physics*, 227(6):3358 – 3382, 2008.
- [59] Ashley J. James and John Lowengrub. A surfactant-conserving volume-of-fluid method for interfacial flows with insoluble surfactant. *Journal of Computational Physics*, 201(2):685 – 722, 2004.

- [60] Jaakko Järvi, Jeremiah Willcock, Howard Hinnant, and Andrew Lumsdaine. *Function Overloading Based on Arbitrary Properties of Types*. *C/C++ Users J.*, 21(6):25–32, June 2003.
- [61] Hrvoje Jasak. *Error Analysis and Estimatio for the Finite Volume Method with Applications to Fluid Flows*. PhD thesis, Imperial College of Science, 1996.
- [62] Hrvoje Jasak and Aleksandar Jemcov. OpenFOAM : A C ++ Library for Complex Physics Simulations ~. In *Int. Work. Coupled Methods Numer. Dyn.*, volume Vol. 1000, pages 1–20, Dubrovnik, Croatia, 2007.
- [63] V. Jayaraman, H. S. Udaykumar, and W. S. Shyy. Adaptive unstructured grid for three-dimensional interface representation. *Numerical Heat Transfer, Part B: Fundamentals*, 32(3):247–265, 1997.
- [64] M. Jemison, E. Loch, M. Sussman, M. Shashkov, M. Arienti, M. Ohta, and Y. Wang. A coupled Level Set-Moment of Fluid method for incompressible two-phase flows. *J. Sci. Comput.*, 54(2-3):454–491, February 2013.
- [65] Lluís Jofre, Oriol Lehmkuhl, Jesús Castro, and Assensi Oliva. A 3-D Volume-of-Fluid advection method based on cell-vertex velocities for unstructured meshes. *Comput. Fluids*, 94:14–29, 2014. ISSN 00457930. doi: 10.1016/j.compfluid.2014.02.001.
- [66] Lluís Jofre, Ricard Borrell, Oriol Lehmkuhl, and Assensi Oliva. Parallel load balancing strategy for Volume-of-Fluid methods on 3-D unstructured meshes. *J. Comput. Phys.*, 282:269–288, 2015. ISSN 0021-9991. doi: 10.1016/j.jcp.2014.11.009. URL <http://dx.doi.org/10.1016/j.jcp.2014.11.009>.
- [67] Nicolai M Josuttis. *The C++ standard library: a tutorial and reference*. Addison-Wesley Professional, 2012. ISBN 978-0321623218.
- [68] Franjo Juretić. *Error Analysis in Finite Volume CFD*. PhD thesis, Imperial College of Science, 2004.
- [69] D.B. Douglas B. Kothe, W.J. WJ Rider, SJ S.J. Mosso, and JS J.S. Brock. Volume tracking of interfaces having surface tension in two and three dimensions. *Aiaa 96-0859*, (January):25, 1996. ISSN 09977546. doi: 10.2514/6.1996-859. URL <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.27.9162>.
- [70] Douglas B. Kothe. Perspective on Eulerian Finite Volume Methods for Incompressible Interfacial Flows. *Free Surf. Flows*, M:267–331, 1999.
- [71] Vincent Le Chenadec and Heinz Pitsch. A 3d unsplit forward/backward volume-of-fluid approach and coupling to the level set method. *Journal of computational physics*, 233:10–33, 2013.
- [72] C Li, S Pion, and C K Yap. Recent progress in exact geometric computation. *J. Log. Algebr. Program.*, 64:85–111, 2005. doi: 10.1016/j.jlap.2004.07.006.

- [73] Petar Liovic, Murray Rudman, Jong Leng Liow, Djamel Lakehal, and Doug Kothe. A 3D unsplit-advection volume tracking algorithm with planarity-preserving interface reconstruction. *Comput. Fluids*, 35(10):1011–1032, 2006. ISSN 00457930. doi: 10.1016/j.compfluid.2005.09.003.
- [74] R. Löhner. Robust, vectorized search algorithms for interpolation on unstructured grids. *Journal of computational Physics*, 118(2):380–387, 1995.
- [75] Joaquin López, J. Hernández, P. Gómez, and F. Faura. A volume of fluid method based on multidimensional advection and spline interface reconstruction. *J. Comput. Phys.*, 195(2):718–742, 2004. ISSN 00219991. doi: 10.1016/j.jcp.2003.10.030.
- [76] J. López and J. Hernández. Analytical and geometrical tools for 3d volume of fluid methods in general grids. *Journal of Computational Physics*, 227(12):5939–5948, 2008.
- [77] J. López, C. Zanzi, P. Gómez, F. Faura, and J. Hernández. A new volume of fluid method in three dimensions—Part II: Piecewise-planar interface reconstruction with cubic-Bézier fit. *International Journal for Numerical Methods in Fluids*, 58(8), 2008.
- [78] T. Marić, H. Marschall, and D. Bothe. voFoam - A geometrical volume of fluid algorithm on arbitrary unstructured meshes with local dynamic adaptive mesh refinement using OpenFOAM. *arXiv preprint arXiv:1305.3417*, 2013.
- [79] T. Marić, H. Marschall, and D. Bothe. An object-oriented validation library for two-phase DNS algorithms. In *Euromech Colloquium 555 on Small-Scale Numerical Methods for Multi-Phase Flows*, Bordeaux, France, August 28–30, 2013.
- [80] T. Marić, H. Marschall, and D. Bothe. On the Adaptive Mesh Refinement for a 3D geometrical Volume-of-Fluid transport algorithm on unstructured meshes using OpenFOAM®. In *Int. Conf. on Multiphase Flow, ICMF*, Jeju, Korea, May 26–31, 2013.
- [81] T. Marić, H. Marschall, and D. Bothe. lentFoam - A hybrid Level Set/Front Tracking method on unstructured meshes. *Comput. Fluids*, 113:20–31, 2015. ISSN 00457930. doi: 10.1016/j.compfluid.2014.12.019.
- [82] Holger Marschall. *Towards the Numerical Simulation of Multi-Scale Two-Phase Flows*. PhD thesis, Technische Universität München, 2011.
- [83] Dimitri J. Mavriplis. Revisiting the Least-squares Procedure for Gradient Reconstruction on Unstructured Meshes. Technical report, National Institute of Aerospace, Hampton, Virginia, 2003.
- [84] Steve McConnell. *Code Complete, Second Edition*. Microsoft Press, 2004. ISBN 978-0735619678.

- [85] D. Meagher. Geometric modeling using octree encoding. *Computer Graphics and Image Processing*, 19(2):129 – 147, 1982.
- [86] Dinesh P Mehta and Sartaj Sahni. *Handbook of data structures and applications*. CRC Press, 2004.
- [87] R. Mittal and G. Iaccarino. Immersed boundary methods. *Annu. Rev. Fluid Mech.*, 37:239–261, 2005.
- [88] S J Mosso, B K Swartz, D B Kothe, and S P Clancy. Recent enhancements of volume tracking algorithms for irregular grids. *Los Alamos Natl. Lab. Los Alamos, NM, LA\_UR\_96\_277*, pages 20–23, 1996.
- [89] S.J. Mosso, B.K. Swartz, D.B. Kothe, and R.C. Ferrell. A Parallel, Volume-Tracking Algorithm for Unstructured Meshes. In P. Schiano, A. Ecer, J. Periaux, and N. Satofuka, editors, *Parallel Comput. Dyn. Algorithms Results Using Adv. Comput.*, number MAY, pages 368–375, Capri, Italy, 1996. ISBN 9780444823274. doi: 10.1016/B978-044482327-4/50113-3. URL <http://linkinghub.elsevier.com/retrieve/pii/B9780444823274501133>.
- [90] Stewart Mosso, Christopher Garasi, and Richard Drake. A smoothed two- and three-dimensional interface reconstruction method. *Computing and visualization in science*, 12(7):365–381, 2009.
- [91] F Moukalled, L Mangani, and M Darwish. *The Finite Volume Method in Computational Fluid Dynamics*. Springer, 2015.
- [92] M. Muradoglu and G. Tryggvason. A Front-Tracking method for computation of interfacial flows with soluble surfactants. *Journal of Computational Physics*, 227(4):2238 – 2262, 2008.
- [93] S Muzaferija and M. Peric. COMPUTATION OF FREE-SURFACE FLOWS USING THE FINITE-VOLUME METHOD AND MOVING GRIDS. *Numer. Heat Transf. Part B Fundam.*, 32(4):369–384, 1997. doi: 10.1080/10407799708915014.
- [94] R. H. Nochetto and S. W. Walker. A hybrid variational front tracking-level set mesh generator for problems exhibiting large deformations and topological changes. *Journal of Computational Physics*, 229(18):6243–6269, 2010.
- [95] W. F. Noh and P. R. Woodward. SLIC (Simple Line Interface Calculation) method. *Proc. Fifth Int. Conf. Numer. Methods Fluid Dyn. June 28–July 2, 1976 Twente Univ. Enschede*, pages 330–340, 1976. doi: 10.1007/3-540-08004-X\_336.
- [96] Joseph O’Rourke. *Computational geometry in C*. Cambridge university press, 1998.
- [97] S. Osher and R. P. Fedkiw. Level Set methods: An overview and some recent results. *Journal of Computational Physics*, 169(2):463 – 502, 2001.

- [98] Mark Owkes and Olivier Desjardins. A computational framework for conservative, three-dimensional, unsplit, geometric transport with application to the volume-of-fluid (VOF) method. *J. Comput. Phys.*, 270: 587–612, 2014. ISSN 10902716. doi: 10.1016/j.jcp.2014.04.022. URL <http://dx.doi.org/10.1016/j.jcp.2014.04.022>.
- [99] Mark Owkes and Olivier Desjardins. A mesh-decoupled height function method for computing interface curvature. *Journal of Computational Physics*, 281:285–300, 2015.
- [100] Peter Pacheco. *An introduction to parallel programming*. Morgan Kaufmann, 2011.
- [101] Suhas Patankar. *Numerical heat transfer and fluid flow*. CRC Press, 1980.
- [102] C. S. Peskin. *Flow patterns around heart valves: a digital computer method for solving the equations of motion*. PhD thesis, Sue Golding Graduate Division of Medical Sciences, Albert Einstein College of Medicine, Yeshiva University, 1972.
- [103] James Edward Pilliod and Elbridge Gerry Puckett. Second-order accurate volume-of-fluid algorithms for tracking material interfaces. *J. Comput. Phys.*, 199(2):465–502, 2004. ISSN 00219991. doi: 10.1016/j.jcp.2003.12.023.
- [104] Stephane Popinet. Gerris: a tree-based adaptive solver for the incompressible Euler equations in complex geometries. *Journal of Computational Physics*, 190(2):572 – 600, 2003.
- [105] Stéphane Popinet. An accurate adaptive solver for surface-tension-driven interfacial flows. *Journal of Computational Physics*, 228(16):5838 – 5866, 2009.
- [106] Elbridge Puckett. On the second-order accuracy of volume-of-fluid interface reconstruction algorithms: convergence in the max norm. *Communications in Applied Mathematics and Computational Science*, 5(1):99–148, 2010.
- [107] Elbridge Puckett. A volume-of-fluid interface reconstruction algorithm that is second-order accurate in the max norm. *Communications in Applied Mathematics and Computational Science*, 5(2):199–220, 2010.
- [108] Elbridge Puckett. Second-order accuracy of volume-of-fluid interface reconstruction algorithms, ii: An improved constraint on the cell size. *Communications in Applied Mathematics and Computational Science*, 8(1):123–158, 2014.
- [109] R. J. Leveque. High-resolution conservative algorithms for incompressible flow. *SIAM J. Numer. Anal.*, 33(2):627–665, 1996. ISSN 0036-1429. doi: 10.1137/0733033.

- [110] Yuriko Renardy and Michael Renardy. PROST: A Parabolic Reconstruction of Surface Tension for the Volume-of-Fluid Method. *Journal of Computational Physics*, 183(2):400 – 421, 2002.
- [111] William J Rider and Douglas B Kothe. Reconstructing Volume Tracking. *J. Comput. Phys.*, 141(2):112–152, 1998. ISSN 00219991. doi: 10.1006/jcph.1998.5906. URL <http://www.sciencedirect.com/science/article/pii/S002199919895906X>~~\$\backslash\$delimiter~~"026E30F\$nh<http://dx.doi.org/10.1006/jcph.1998.5906>.
- [112] Allen C. Robinson, Jay Mosso, Chris Siefert, Jonathan Hu, Tom Gardiner, and Joe Crepeau. Consequences for scalability arising from multi-material modeling. Numerical methods for multi-material fluid flows, Czech Technical University, Prague, Czech Republic, 2007. URL [http://www-troja.fjfi.cvut.cz/~multimat07/presentations/tuesday/Robinson/Robinson\\_scalability.pdf](http://www-troja.fjfi.cvut.cz/~multimat07/presentations/tuesday/Robinson/Robinson_scalability.pdf).
- [113] Johan Roenby, Henrik Bredmose, and Hrvoje Jasak. A Computational Method for Sharp Interface Advection. *R. Soc. Open Sci.*, 3(11), 2016. doi: 10.1098/rsos.160405. URL <http://arxiv.org/abs/1601.05392>.
- [114] I. Roghair, J. Martínez Mercado, M. Van Sint Annaland, H. Kuipers, C. Sun, and D. Lohse. Energy spectra and bubble velocity distributions in pseudo-turbulence: Numerical simulations vs. experiments. *International Journal of Multiphase Flow*, 37(9):1093 – 1098, 2011.
- [115] Jim Rupert and Raimund Seidel. On the Difficulty of Tetrahedralizing 3-Dimensional Non-Convex Polyhedra. In *SCG '89 Proc. fifth Annu. Symp. Comput. Geom.*, pages 380–392, 1989. ISBN 0897913183. doi: 10.1145/73833.73875.
- [116] Yousef Saad. *Iterative methods for sparse linear systems*. Siam, 2003.
- [117] Tim Sackton. Photograph, still as rain [42/366]. <https://www.flickr.com/photos/sackton/24602119749/>. Accessed: 2017-04-14.
- [118] H. Samet. *The design and analysis of spatial data structures*, volume 199. Addison-Wesley Reading, MA, 1990.
- [119] Ruben Scardovelli and Stéphane Zaleski. Direct Numerical Simulation of Free-Surface and Interfacial Flow. *Annu. Rev. Fluid Mech.*, 31(1):567–603, 1999. ISSN 0066-4189. doi: 10.1146/annurev.fluid.31.1.567.
- [120] Ruben Scardovelli and Stephane Zaleski. Interface reconstruction with least-square fit and split Eulerian-Lagrangian advection. *Int. J. Numer. Methods Fluids*, 41(3):251–274, 2003. ISSN 02712091. doi: 10.1002/fld.431.
- [121] Schneider. *Geometric Tools for Computer Graphics*. Morgan Kaufmann, October 2002. ISBN 1-55860-594-0.

- [122] J.A. Sethian. *Level Set Methods and Fast Marching Methods*, volume 3. Cambridge university press, 1999.
- [123] James A Sethian. A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences*, 93(4): 1591–1595, 1996.
- [124] Jonathan Richard Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, Carnegie-Mellon University. Department of Computer Science, 1994.
- [125] S. Shin and D. Juric. Modeling three-dimensional multiphase flow using a level contour reconstruction method for Front Tracking without connectivity. *Journal of Computational Physics*, 180(2):427 – 470, 2002.
- [126] S. Shin and D. Juric. High order level contour reconstruction method. *Journal of Mechanical Science and Technology*, 21(2):311–326, 2007.
- [127] S. Shin and D. Juric. A hybrid interface method for three-dimensional multiphase flows based on front tracking and level set techniques. *International Journal for Numerical Methods in Fluids*, 60(7):753–778, 2009.
- [128] Seungwon Shin and Damir Juric. A hybrid interface method for three-dimensional multiphase flows based on front tracking and level set techniques. *International Journal for Numerical Methods in Fluids*, 60:753–778, 2009.
- [129] Seungwon Shin, Jalel Chergui, and Damir Juric. A solver for massively parallel direct numerical simulation of three-dimensional multiphase flows. *arXiv preprint arXiv:1410.8568*, 2014.
- [130] Bjarne Stroustrup. *The C++ programming language*. Pearson Education, 2013. ISBN 978-0321563842.
- [131] M. Sussman and E. G. Puckett. A coupled Level Set and Volume-of-Fluid method for computing 3d and axisymmetric incompressible two-phase flows. *Journal of Computational Physics*, 162(2):301 – 337, 2000.
- [132] M. Sussman, E. Fatemi, P. Smereka, and S. Osher. An improved level set method for incompressible two-phase flows. *Computers & Fluids*, 27(5–6): 663 – 680, 1998.
- [133] M. Sussman, A. S. Almgren, J. B. Bell, P. Colella, L. H. Howell, and M. L. Welcome. An Adaptive Level Set Approach for Incompressible Two-Phase Flows. *Journal of Computational Physics*, 148(1):81 – 124, 1999.
- [134] Mark Sussman, Emad Fatemi, Peter Smereka, and Stanley Osher. AN IMPROVED LEVEL SET METHOD FOR INCOMPRESSIBLE TWO-PHASE FLOWS. *Comput. Fluids*, 27(97):663–680, 1998.
- [135] Blair Swartz. The second-order sharpening of blurred smooth borders. *Mathematics of Computation*, 52(186):675–714, 1989.



- [136] T. Tolle, T. Marić, H. Marschall, and D. Bothe. Extending the hybrid Level Set / Front Tracking (LENT) method on unstructured meshes for surface tension driven flows. *"In preparation."*, 2017.
- [137] G. M. Treece, R. W. Prager, and A. H. Gee. Regularised Marching Tetrahedra: Improved Iso-Surface Extraction. *Computers and Graphics*, 23:583–598, 1998.
- [138] G. Tryggvason, B. Bunner, A. Esmaeeli, D. Juric, N. Al-Rawahi, W. Tauber, J. Han, S. Nas, and Y.-J. Jan. A Front-Tracking method for the computations of multiphase flow. *Journal of Computational Physics*, 169(2):708 – 759, 2001.
- [139] G. Tryggvason, R. Scardovelli, and S. Zaleski. *Direct Numerical Simulations of Gas-Liquid Multiphase Flows*. Cambridge University Press, 2011. ISBN 9780521782401.
- [140] Ž. Tuković and H. Jasak. A moving mesh finite volume interface tracking method for surface tension dominated interfacial fluid flow. *Comput. Fluids*, 55:70–84, 2012. ISSN 00457930. doi: 10.1016/j.compfluid.2011.11.003. URL <http://www.sciencedirect.com/science/article/pii/S0045793011003380>.
- [141] Zeljko Tuković. *Metoda kontrolnih volumena na domenama promjenjivog oblika*. PhD thesis, Sveučiliste u Zagrebu, Fakultet Strojarsstva i Brodogradnje, 2005.
- [142] Onno Ubbink. *Numerical prediction of two fluid systems with sharp interfaces*. Imperial College of Science, Technology & Medicine, 1997.
- [143] S. O. Unverdi and G. Tryggvason. A front-tracking method for viscous, incompressible, multi-fluid flows. *Journal of Computational Physics*, 100(1): 25 – 37, 1992.
- [144] David Vandevoorde and Nicolai M. Josuttis. *C++ Templates: The Complete Guide*. Addison-Wesley Professional, 2002. ISBN 978-0201734843.
- [145] Henk Kaarle Versteeg and Weeratunge Malalasekera. *An introduction to computational fluid dynamics: the finite volume method*. Pearson Education, 2007.
- [146] J. A. C. Weideman. Numerical Integration of Periodic Functions : A Few Examples. *Am. Math. Mon.*, 109(1):21–36, 2002.
- [147] H. G. Weller, G. Tabor, H. Jasak, and C. Fureby. A tensorial approach to computational continuum mechanics using object-oriented techniques. *Comput. Phys.*, 12(6):620–631, 1998.
- [148] David L Youngs. Time-dependent multi-material flow with large fluid distortion. *Numerical methods for fluid dynamics*, 24:273–285, 1982.



- [149] Steven T. Zalesak. Fully Multidimensional Flux-Corrected Transport Algorithm For Fluids. *J. Comput. Phys.*, 362:335–362, 1979.
- [150] J. Zhang, D.M. Eckmann, and P.S. Ayyaswamy. A front tracking method for a deformable intravascular bubble in a tube with soluble surfactant transport. *Journal of Computational Physics*, 214:366 – 396, 2006.
- [151] Q. Zhang. On a Family of Unsplit Advection Algorithms for Volume-of-Fluid Methods. *SIAM J. Numer. Anal.*, 51(5):2822–2850, 2013. URL <http://epubs.siam.org/doi/abs/10.1137/120897882>  
<http://epubs.siam.org/doi/pdf/10.1137/120897882>.
- [152] Qinghai Zhang. On generalized donating regions: Classifying Lagrangian fluxing particles through a fixed curve in the plane. *J. Math. Anal. Appl.*, 424(2):861–877, 2015. ISSN 10960813. doi: 10.1016/j.jmaa.2014.11.043.
- [153] Qinghai Zhang and Aaron Fogelson. Fourth-Order Interface Tracking in Two Dimensions via an Improved Polygonal Area Mapping Method. *SIAM J. Sci. Comput.*, 36(5):2369–2400, oct 2014. ISSN 1064-8275. doi: 10.1137/140951886. URL <http://epubs.siam.org/doi/abs/10.1137/140951886>.
- [154] Qinghai Zhang and Philip L.-F. Liu. A new interface tracking method: The polygonal area mapping method. *Journal of Computational Physics*, 227(8): 4063 – 4088, 2008.



---

## INDEX

---

- advected phase centroid, 46
- around the corner flux, 57
- atomic operations, 133
  
- barycentric triangulation, 96
- Boundary condition discretization, 17
- bulk cells, 79
  
- cap polygon, 115
- cell sphere, 81
- cell sphere radius, 81
- cell sphere triangles, 81
- Central differencing scheme, 16
- circular shift, 10
- computational mesh, mesh, 9
- concept, 126
- convex stencil, 41
- cross stencil, 41
- cut-out slab, slab, 50
  
- defined donating region, 57
- dimensionally split, 21
- dimensionally split algorithm, 52
- dimensionally unsplit, 21
- discrete Lagrangian trajectory, 23
- discrete solution domain, 9
- Divergence theorem discretization, 14
- donating region, 57
  
- edge triangulation, 97
- Eulerian flux-based method, 21
- exact interface, 28
  
- face, 10
- face volume fraction, 84
- face-point-faces, 121
- finite volume, 9
- finite volume face, face, 10
- flux polyhedron, 32
- flux stencil, 93
- flux triangulation, 97
- flux volume, 23, 32
- flux volume correction, 108
- flux volume overlap, 53
- flux volume scaling coefficient, 63
- flux volume triangulation, 32
- front, 145, 149
- front triangle, 149
- front triangle vertex, 150
  
- Gauss divergence theorem, 13
- geometric predicate, 112
- geometrical diffusion, 37
- geometrical flux volume, 32
  
- implicit Euler scheme, 14
- Implicit Euler temporal discretization, 15
- implicit interface, 126
- indirect addressing, 10, 149
- indirect indexed access, 10
- indirect indexing format, 10
- initial interface, 75
- inside cells, 79
- interface cell, 24
- interface cells, 25, 75, 87
- interface mesh, 28, 75
- interface orientation, 30
- interface polygon, 42
- interface positioning, 30
- interface reconstruction, 27
- interleaving communication and computation, 74
- intersection halfspace, 109
- iterative flux correction, 106
  
- jetsam, flotsam, 37
  
- Lagrangian backward tracing, 90
- Lagrangian tracking / Eulerian re-mapping method, 21
  
- manifold mesh, 145
- material volume, 67

- mesh cell, cell, 9
- mesh edges, 43
- mesh points, 10
- narrow band, 93
- narrow band face, 93
- nearest triangle, 81
- operator splitting algorithm, 52
- oriented triangulation, 96
- overshoot, 53, 119
- owner-neighbor addressing, 16
- parallel speedup, 179
- parallelization, method parallelization, 74
- phase flux volume, 23
- phase flux volume contributions, 26
- piecewise constant interface, 37
- piecewise linear interface, 37
- point cell inverse distance weight, 84
- point cells, 84
- point volume, 26
- point volume fraction, 43, 83
- point-in-polyhedron algorithm, 75
- points of variation, 131
- polygon soup, 115
- positive halfspace, 25
- pyramid flux correction, 107
- ray-crossing algorithm, 75
- reconstructed phase centroid, 46
- reconstruction correctness, 58
- reconstruction tolerance, 30
- ruled surface, 23
- signed distance diffusion coefficient, 81
- software design scaling, 131
- Source term, 15
- Source term linearisation, 15
- squared search distance, 79
- Steiner point, 100
- stream scheme, 55
- stream tube, 55
- surface area normal vector, 11
- surface average, 12
- surface linearization, 24
- surface triangulation, 24
- Taylor series point displacement, 92
- template argument, 126
- time step, 12
- tolerance-based filtering, 113
- trapezoidal point displacement, 92
- trapezoidal rule, 12
- triangle neighborhood, 149
- triangle normal area vector, 150
- triangle-cell map, 161
- type lifting, 126
- type traits, 127
- undershoot, 53, 119
- unit operation, 78
- visible triangles, 101
- volume average, 12
- volume conservation correction, 32
- volume fraction advection, 31
- volume fraction pre-processing, 28
- wisp cells, 61
- wisp tolerance, 121
- wisps, 56, 106, 119

# TOMISLAV MARIĆ

## WORK EXPERIENCE

*Apr 2014 – Dec 2017* Co-founder, sourceflux UG

Development of OpenFOAM®<sup>1</sup> projects involving numerical simulations and software development, preparing and giving OpenFOAM courses for industrial and academic clients.

*Jul 2016 – Dec 2017* Research Scientist, Mathematical Modeling and Analysis, TU Darmstadt

High Performance Computing aspects and parallelisation of Lagrangian / Eulerian two-phase flow methods using OpenMP and Open MPI as well as the accurate numerical modeling of surface tension forces. Agile software design principles and version control for the Collaborative Research Center 1194 using Atlassian JIRA, Confluence, and Bitbucket.

*Apr 2011 – Jul 2016* Research Assistant, Mathematical Modeling and Analysis, TU Darmstadt

Developing Lagrangian/Eulerian methods for Direct Numerical Simulations of two-phase flows using the C++ programming language, within the OpenFOAM open source framework for Computational Fluid Dynamics.

## EDUCATION

*Apr 2011 – Nov 2017* Dr.-Ing, Mathematics department, TU Darmstadt

Ph.D. Thesis: *Lagrangian / Eulerian numerical methods for fluid interface advection on unstructured meshes*

Description: Developed and implemented two methods for accurate tracking of fluid interfaces (sharp discontinuities) that undergo strong deformation, breakup and coalescence.

*Oct 2008 – Nov 2010* M.Sc., Mechanical engineering department, University of Zagreb / Technical University of Munich

GPA: 5.0 / 5.0 · Specialization: Numerical Simulations

Thesis: *Development and Implementation of a Volume-of-Fluid Method with Geometrical Interface Reconstruction for Incompressible Free-Surface Flows using OpenFOAM*

Description: Development of a geometrical algorithm for reconstructing the boundary between two immiscible fluid phases on unstructured meshes.

*Sept 2004 – Sept 2008* Bachelor, Mechanical engineering department, University of Zagreb

GPA: 4.63 / 5.0 · Specialization: Numerical Simulations

Thesis: *Numerical investigation of a 3D droplet impact onto an inclined surface using OpenFOAM*

Description: Focus on programming field processing applications.

---

<sup>1</sup> OPENFOAM® is a registered trade mark of OpenCFD Limited, producer and distributor of the OpenFOAM software via [www.openfoam.com](http://www.openfoam.com).

## PUBLICATIONS

## Books

*Sep 2014*      The OpenFOAM Technology Primer

Authors: Tomislav MARIĆ, Jens HÖPKEN, Kyle MOONEY

## Articles

*Jan 2014*      lentFoam - A hybrid Level Set / Front Tracking method on unstructured meshes

Marić, T., Marschall, H. and Bothe, D., 2015. lentFoam-A hybrid Level Set/Front Tracking method on unstructured meshes. *Computers & Fluids*, 113, pp.20-31.

*May 2013*      voFoam - A geometrical Volume of Fluid algorithm on arbitrary unstructured meshes with local dynamic adaptive mesh refinement using OpenFOAM

Maric, T., Marschall, H. and Bothe, D., 2013. voFoam-a geometrical volume of fluid algorithm on arbitrary unstructured meshes with local dynamic adaptive mesh refinement using OpenFOAM. arXiv preprint arXiv:1305.3417.

## Conference Articles

*May 2013*      International Conference on Multiphase Flows

T. Marić, H. Marschall, and D. Bothe. On the Adaptive Mesh Refinement for a 3D geometrical Volume-of-Fluid transport algorithm on unstructured meshes using OpenFOAM®. In *Int. Conf. on Multiphase Flow, ICMF*, Jeju, Korea, May 26-31, 2013

*Aug 2013*      Euromech Colloquium 555 on Small-Scale Numerical Methods for Multi-Phase Flows

T. Marić, H. Marschall, and D. Bothe. An object-oriented validation library for two-phase DNS algorithms. In *Euromech Colloquium 555 on Small-Scale Numerical Methods for Multi-Phase Flows*, Bordeaux, France, August 28–30, 2013

## AWARDS AND SCHOLARSHIPS

2009/2010 · Technical University of Munich, Exchange Scholarship

2008/2009 · University of Zagreb, scholarship for best students

2005 · FAMENA, University of Zagreb, "Davorin Bazjanac" best GPA award

December 14, 2017